Conception d'architectures systoliques integrées

Dominique Lavenier

HABILITATION A DIRIGER DES RECHERCHES

présentée devant

L'Université de Rennes 1
Institut de Formation Supérieure
en Informatique et en Communication

par

Dominique LAVENIER

CONCEPTION D'ARCHITECTURES SYSTOLIQUES INTEGREES

soutenue le 1er octobre 1997 devant le jury composé de

M.	J. Lenfant	Président
MM.	D. Litaize	Rapporteurs
	J. Vuillemin	
	M. Weinfeld	
MM.	J.P. Banâtre	Examinateurs
	P. Frison	
	P. Quinton	

Remerciements

Je remercie Jacques Lenfant, président de l'Université de Rennes 1 et architecte de machines informatiques, qui me fait l'honneur de présider le jury.

Je remercie également Daniel Litaize, Professeur à l'Université Paul Sabatier de Toulouse, Jean Vuillemin, Professeur à l'ENS, et Michel Weinfeld, directeur de recherches au CNRS qui m'ont fait l'amitié d'être à la fois rapporteurs et membres du jury.

Je remercie Jean Pierre Banâtre, directeur de l'IRISA, pour son soutien permanent au cours de ces quelques années de recherche.

Je tiens tout particulièrement à remercier Patrice Quinton, responsable du projet API, qui m'a toujours laissé une grande liberté dans mes activités de recherche, tout en me prodiguant une multitude de conseils et de suggestions, tous aussi pertinents qu'efficaces.

J'exprime mes plus chaleureux remerciements à mes deux ex-collègues Patrice Frison et Eric Gautrin, tous deux exilés maintenant, l'un à Vannes, l'autre à l'Atelier, pour leur active collaboration lors de mes débuts dans le métier de chercheur. Tous les travaux présentés dans ce document tirent leurs origines dans cette coopération initiale.

Je remercie, bien sûr, l'ensemble des membres – passés et présents – de l'équipe API, et plus particulièrement ceux avec qui mes collaborations ont été les plus étroites. Que Frédéric Raimbault, Roderick MacConnell, Marc Vieillot, Pascale Guerdoux-Jamet et Charles Wagner trouvent ici toute ma reconnaissance.

J'adresse enfin mes plus sincères remerciements, d'une part, à Bernard Pottier – et au passage à toute l'équipe Brestoise – et, d'autre part, à Jean Vuillemin pour m'avoir initiés au monde de la logique reconfigurable. Sans les projets ArMen et PAM, mon intérêt pour ce domaine ne serait sans doute pas ce qu'il est aujourd'hui.

à Domi,
Anaëlle et Loïc

Bien sûr, aucun chercheur n'est responsable,
à lui tout seul, d'une idée scientifique.
Chaque idée nouvelle est en partie
une synthèse de nombreuses observations
et hypothèses plus anciennes,
et il est toujours possible de trouver
des signes d'une intuition nouvelle
dans le travail qui a précédé.

Christopher Wills. La sagesse des gènes.

Table des Matières

1
Introduction

Ce document synthétise huit années de recherche. Son but est d'en présenter une vue générale en répartissant mes activités par thème et en montrant les liens qui les unissent.

Mon thème de recherche principal est la conception d'architectures systoliques intégrées (VLSI[1] et/ou FPGA[2]) et s'appuie sur une domaine applicatif particulier : le traitement des séquences (correction orthographique et biologie moléculaire). Ces travaux de recherche se sont déroulés à l'IRISA, dans l'équipe API dirigée par Patrice Quinton.

Une architecture systolique intégrée est un réseau régulier de processeurs identiques, connectés localement et dédiés à un traitement particulier[3]. L'ensemble est implanté sur un même support de silicium et fonctionne de manière synchrone, au rythme d'une horloge globale.

La conception de tels systèmes s'appuie avant tout sur la régularité des traitements qui, en général, s'expriment de manière très synthétique. Ainsi, à partir de spécifications de haut niveau, de nombreuses équipes de recherche visent à dériver automatiquement des architectures parallèles spécialisées. L'objectif est de produire un système intégré (un composant VLSI) sans intervention manuelle, ou du moins, sans que l'intervention soit source d'erreurs. L'automatisation garantit à la fois un système correct et un temps de conception court.

La conception automatique d'architectures régulières est l'objectif principal de l'équipe API, mais pas le seul. Une seconde activité, complémentaire de la première et initialisée par Patrice Frison, conçoit et réalise des prototypes de machines systoliques. Les enseignements tirés des réalisations sont reportés

[1] Very Large Scale Integration

[2] Field Programmable Gate Array

[3] Cette définition exclut les architectures systoliques programmables telles que l'accélérateur MicMacs [54] ou le réseau iWarp [7]

au niveau méthodologique, tandis que les outils et les méthodes de conception imaginés sont expérimentés sur les applications. Ce va et vient continuel entre stratégies de conception et réalisation de prototypes garantit l'élaboration d'outils et de méthodes cohérentes et efficaces : on ne peut développer correctement des outils de synthèse ou de CAO sans une expérience réelle en matière de conception d'architectures de machines et de circuits VLSI.

Plus précisément, les travaux de l'équipe API sont structurés suivant trois grandes thématiques :

- la définition de méthodologies de conception : il s'agit d'étudier des stratégies qui, à partir d'une spécification, proposent un enchaînement de tâches pour aboutir au dessin de masque du circuit ;
- le développement d'outils : cet axe supporte les méthodologies proposées ;
- la réalisation de machines : cette activité valide les méthodes et les outils.

Mes recherches abordent ces trois axes et s'appuient principalement sur un domaine d'applications spécifique, le traitement des séquences. A travers la réalisation de différentes machines, circuits et outils (SAMBA, API69, C-stolic) plusieurs approches de conception ont été testées en essayant, dans la mesure du possible, d'utiliser les outils développés en interne.

Bien évidemment, de nombreuses personnes m'ont aidé dans ces travaux, notamment les étudiants que j'ai pu encadrer pendant leur de stage de DEA et/ou leur thèse. Au fur et à mesure du document, et en fonction des différents projets, j'ai indiqué leur contribution.

Même si mes travaux de recherche m'ont conduit à ne considérer qu'une classe restreinte de machines (les architectures systoliques), les méthodes, les choix ou les stratégies de conception qui ont pu être imaginés couvrent un spectre d'activités beaucoup plus large. En effet, dès lors que l'on est confronté à la conception d'une machine informatique, se pose un certain nombre de problèmes fondamentaux qui doivent être résolus, quelle que soit l'architecture envisagée. Les machines spécialisées n'échappent pas à cette règle. Au contraire, elles exacerbent certains côtés qui rendent difficile, par exemple, l'équilibrage d'une architecture complète. Typiquement, dans le cadre des systèmes systoliques, la puissance crête développée par le réseau, n'est significative que si l'environnement qui l'accueil est adapté

à son rythme. Autrement dit, la même attention doit être portée à tous les éléments du système de manière à produire un ensemble cohérent : les performances globales du système sont dictées par l'élément le plus lent !

Les problèmes d'interfaçage, d'équilibrage, de partitionnement ou de co-design ont donc été au coeur de mes préoccupations. Ils constituent un secteur d'activités qui dépasse largement le cadre de la conception d'architectures systoliques. Les solutions architecturales imaginées, et présentées par la suite, pour tenter de résoudre ces difficultés se situent au delà de cette seule gamme de machines et les résultats peuvent être exploités dans bien d'autres situations.

Le but de ce document n'est pas de décrire en détail l'ensemble de mes travaux de recherche, mais de présenter ceux qui me semblent les plus significatifs et qui synthétisent au mieux mes activités de chercheur dans ce domaine.

Trois réalisations sont présentées et, à travers elles, les méthodologies de conception sous-jacentes. La première concerne une architecture systolique dédiée à la correction orthographique : le circuit API69. L'originalité du travail porte sur la synthèse physique du circuit. La seconde propose un langage (le langage C-stolic) pour la simulation des architectures systoliques. Enfin, la troisième réalisation, la machine SAMBA, est un système dédié à l'analyse des banques de séquences biologiques. Elle a constitué un support très concret pour tester une méthodologie de conception qui s'appuie principalement sur le langage C-stolic.

Avant d'aborder chacune de ces études, un chapitre introductif situe le cadre de mes travaux. Ensuite, les trois chapitres suivants abordent respectivement la conception du circuit API69, l'élaboration du langage C-stolic et le design de la machine SAMBA. Cette présentation correspond à l'ordre chronologique dans lequel se sont déroulés les événements. Le dernier chapitre établit un bilan et dresse quelques perspectives de recherche.

2

Conception d'architectures systoliques intégrées

Cette première partie expose brièvement l'état des recherches dans le domaine de la conception des architectures systoliques intégrées. Nous distinguons trois axes importants : la synthèse, la simulation et le placement/routage. Ce tour d'horizon a pour but de mieux situer nos travaux.

2.1 Introduction

La conception d'un circuit intégré, quel qu'il soit, est un enchaînement de tâches qui, à partir d'une spécification, délivre les masques nécessaires à la fabrication de la puce. La conception d'une architecture systolique intégrée suit les mêmes règles. Elle se différencie, cependant, par le fait que la description initiale du problème peut être très concise et que l'objectif final vise l'obtention d'une structure physique régulière.

Cette notion de régularité est très importante. Les motivations qui conduisent à la considérer sont de deux ordres : l'optimisation du silicium et la simplification de la conception.

A notre connaissance, il n'existe pas d'outil de CAO commercial spécifiquement dédié à la production de tels composants, c'est-à-dire un outil qui, à partir d'une description de haut niveau, tienne compte de la régularité à tous les stades de la conception et produise un dessin de masques régulier. L'explication réside sans doute dans la faible part de marché que représente ce type d'architecture parmi l'ensemble des circuits réalisés ; d'autre part, les outils de CAO existants empêchent nullement de concevoir ce type de structure : les propriétés de régularité ne sont alors pas automatiquement prises en compte et le travail reste à la charge du concepteur. Enfin, la synthèse au-

tomatique de circuits réguliers comporte encore des étapes non entièrement résolues de manière satisfaisante.

Aussi, la conception automatique d'architectures systoliques intégrées est-elle encore du domaine de la recherche. A ce titre, de nombreuses équipes universitaires ont développé leurs propres outils pour expérimenter et valider diverses méthodologies. Ces outils complètent les chaînes de CAO traditionnelles et se greffent à différents niveaux en fonction de leurs particularités.

On peut distinguer trois grandes catégories d'outils qui essaient de tirer parti des propriétés de régularité des architectures systoliques :

- les outils de synthèse ;
- les outils de simulation ;
- les outils de placement/routage.

Les trois paragraphes suivants passent rapidement en revue les travaux réalisés dans chacun de ces domaines.

2.2 La synthèse

Les premiers travaux sur la synthèse de structures régulières sont apparus peu après la naissance du concept *d'architecture systolique*, tel qu'il a été défini par H.T. Kung [50]. Les premiers à s'être intéressés spécifiquement à la synthèse d'architectures systoliques sont Cappello et Steiglitz [11], Moldovan [76], Miranker et Winkler [75] et Quinton [83]. Depuis, beaucoup d'autres travaux ont été consacrés à ce sujet [15, 21, 51, 67, 92, 90].

Ce thème de recherche est, en fait, très proche d'un autre domaine très actif : la parallélisation automatique des boucles dans les programmes séquentiels. Il est bien connu qu'une grande partie du temps d'exécution des programmes scientifiques est passé dans ces portions de code. Aussi, réduire le temps de calcul engendré par ces boucles a des répercutions directes sur le temps de calcul global. L'enjeu est donc d'extraire de ces nids de boucles le parallélisme potentiel pour en dériver un code exécutable sur une machine parallèle. Le parallélisme peut s'exprimer sous forme d'équations récurrentes, équations qui servent ensuite de support pour dériver un code adéquat.

C'est à ce stade que les deux domaines de recherche se rejoignent puisque le point de départ de la synthèse d'architectures systoliques est un système

d'équations récurrentes et que leurs manipulations partagent les mêmes objets mathématiques (polyèdres convexes) et les mêmes techniques d'optimisation (programmation linéaire).

A partir de ces systèmes d'équations récurrentes, la synthèse d'architectures systoliques consiste essentiellement à assigner à chaque calcul un temps et un lieu, de telle manière que les dépendances deviennent locales et qu'elles soient interprétables comme des connexions dédiées entre processeurs. Le but de ce paragraphe n'est pas d'exposer ces technique : le lecteur intéressé peut se référer à l'ouvrage [84] (chapitre 12 et 13) où sont décrites les différentes étapes de la méthode de synthèse par analyse des dépendances.

Ces méthodes et ces techniques de synthèse ont bien sûr donné lieu à des outils permettant de dériver des architectures systoliques. Citons, entre autres, les logiciels HiFi (Hierarchical Interactif Flowgraph Integration) [20] [45], COMPAR (COmpiler for Massively Parallel ARchitectures) [99], CATHEDRAL IV [12], SYS3 (SYstolic SYnthesis SYstem) [43], et les langages CRYSTAL [16] et ALPHA [69].

De l'étude de ces différents systèmes, il ressort que l'accent est surtout mis sur la phase de synthèse architecturale. Peu d'information est disponible, ensuite, pour établir, d'un point de vue physique, une topologie régulière. En fait, le but de ces outils n'est pas forcément d'aboutir à une mise en œuvre VLSI mais, par exemple, de proposer une implantation sur une structure parallèle programmable. Notons, cependant que SYS3 délivre une description symbolique du plan de masse du réseau, et que les recherches menées autour du langage ALPHA visent à dériver une structure régulière en une description de plus bas niveau (AlpHard) où toutes les informations relatives à la régularité sont conservées en vue d'une implémentation VLSI [97].

Si l'usage de ces outils ne délivre pas (encore) directement les masques des circuits intégrés, ils sont néanmoins d'une aide précieuse pour étudier diverses possibilités de "systolisation".

2.3 La simulation

Dans la conception d'un circuit intégré, la simulation tient une très grande place. Elle a deux objectifs principaux : d'une part, elle assure que le fonctionnement du circuit est conforme au cahier des charges ; d'autre part, elle élimine les erreurs de conception.

Dans un contexte de synthèse automatique, il est clair que le second objectif perd de son intérêt : le circuit étant correct par construction, des simulations pour détecter ce type d'erreurs deviennent inutiles. Mais les outils de synthèse sont encore, pour la plupart, à un stade de recherche, et malgré leur évolution, il s'avère que la conception d'un VLSI – une architecture systolique y compris – ne peut, à l'heure actuelle, se passer de simulations. De plus, ces outils ne sont pas parfaits et la simulation reste une manière d'augmenter la confiance dans l'architecture que l'on conçoit. Ajoutons enfin que l'on peut être amené à modifier le résultat de la synthèse, pour des raisons d'optimisation par exemple, et que dans ce cas, on n'échappe pas à une vérification par le biais de simulations.

Il y a plusieurs niveaux de simulation. Le premier est la simulation comportementale où on vérifie la fonctionnalité du circuit. Le dernier niveau est une simulation électrique qui teste si les contraintes de vitesse, de consommation, etc, sont respectées. Entre ces deux niveaux on a recours à des étapes de simulation intermédiaires. Typiquement, si on utilise un langage de description matériel (VHDL, par exemple), la première description se borne à définir la fonctionnalité du circuit en terme de blocs définis à l'aide de primitives de haut niveau. Puis, progressivement, les blocs sont décrits plus précisément pour tendre vers une description proche du matériel. A chaque étape, on vérifie par simulation que les transformations effectuées sont cohérentes, c'est-à-dire qu'elles n'induisent pas un comportement différent du système.

En règle générale, plus la simulation est fine – i.e. plus on se rapproche d'une description matérielle – plus elle est longue. Pour des circuits complexes, le temps total consacré à la simulation peut représenter des centaines, voire de milliers d'heures de calcul. Il est donc normal de chercher à réduire ce temps.

La simulation comportementale d'architectures systoliques se tourne naturellement vers les machines parallèles et les langages associées. L'architecture du réseau systolique étant supposée connue (déduite à partir d'outils de synthèse, ou par tout autre moyen), la programmation consiste à décrire l'action d'une cellule de base, ses interactions avec les cellules voisines et, éventuellement, les échanges avec l'extérieur.

Bon nombre de langages permettent ce type de description. F. Raimbault, dans sa thèse [89] (chapitre 1.3), en fait une analyse détaillée. A titre d'exemples, citons les langages à parallélisme de contrôle : le langage OCCAM [78] conçu pour la programmation des Transputers, les langages W2 [52] et IWCC [35] développés respectivement pour les machines WARP et iWARP ; et les

langages à parallélisme de données : le langage ACTUS [82], un des premiers langages de ce type dédiés à la programmation des machines SIMD telles que l'ILLIAC IV et le CRAY-1, le langage POMPC [79] qui contient à la fois les mécanismes présents dans C* [95] (Connection Machine) et MPL [18] (MasPar), ou encore, plus récemment le langage NSL (B-SYS) [46], et sans oublier, bien sûr, C-stolic [87].

Tous ces langages n'ont pas été conçus spécifiquement pour supporter le modèle systolique ; certains possèdent un cadre de programmation plus étendu dans lequel le modèle systolique s'insère. L'efficacité dépend alors de l'adéquation de ces langages à exprimer le modèle systolique et de l'adéquation des machines parallèles à exécuter ces langages. L'accélération des simulations peut donc être faible, voire inexistante (par rapport à une mise en œuvre séquentielle). A ce niveau, ceci ne constitue pas véritablement un handicap : en général, le temps de calcul de la simulation comportementale représente un faible pourcentage par rapport au temps de simulation total. Cette étape valide plutôt la parallélisation.

En revanche, les simulations suivantes (niveau portes, en particulier) exigent d'autres techniques pour procurer une accélération significative. Depuis quelques années, l'évolution des technologies FPGA a permis l'essor d'une approche relativement performante : l'usage d'accélérateurs matériels reconfigurables. Les machines des sociétés Meta-Systems [9] et QuickTurn [85] sont de bons exemples : le code HDL[1] est directement émulé sur un support matériel (FPGA). La parallélisation est ainsi réalisée à un grain très fin et est extrêmement performante, le facteur d'accélération (d'après les fabricants) se situant entre 10 000 et 1 000 000 par rapport à un simulateur classique mis en œuvre sur une machine séquentielle !

Ces chiffres impressionnants doivent cependant être modulés par le fait qu'entre la description HDL et l'implantation physique du circuit sur la structure FPGA, il peut s'écouler un temps relativement long dû au processus de compilation d'un code HDL en code FPGA, contrairement à la simulation sur une machine programmable qui débute quasi instantanément. Ainsi, une simulation peut avoir demandé plusieurs heures de préparation ! Les cycles de correction-vérification – incontournables pendant les phases de mise au point – sont alors aussi importants qu'avec une approche classique. Les outils de placement/routage des composants FPGA sont principalement responsables de cette latence.

[1] HDL: Hardware Description Language

Les outils qui effectuent la transformation du code HDL en code FPGA font abstraction de l'architecture à simuler ; ils ne prennent pas en compte d'éventuelles régularités qui pourraient être exploitées pour minimiser la phase de compilation : les accélérateurs étant constitués d'un assemblage *régulier* de composants FPGA, l'implémentation d'une architecture systolique s'y prête pourtant naturellement bien. Il s'agirait d'établir la configuration d'un seul FPGA et de la reporter à l'ensemble des composants. Le gain sur le temps de simulation (cycle de correction-vérification) ne porterait donc pas sur l'implémentation matérielle, mais sur la manière dont cette implémentation est réalisée.

En résumé, la simulation comportementale d'une architecture systolique peut naturellement s'effectuer sur une machine parallèle. L'efficacité est fonction de la double adéquation langage/modèle systolique et machine/langage, mais n'est pas un facteur déterminant à ce stade de la conception. Par contre, l'accélération des simulations de niveau portes nécessite des techniques plus sophistiquées, l'une d'elles étant l'implantation sur FPGA de l'architecture, approche extrêmement performante mais qui nécessite un temps de compilation encore très long. L'exploitation de la régularité des architectures systoliques permettrait de réduire, sans aucun doute, fortement ce temps.

2.4 Le placement/routage

Une des motivations principales dans l'intégration des réseaux systoliques est la conservation, au niveau physique, d'une structure régulière. Elle évite le routage et procure une meilleure intégration tout en garantissant une propagation optimale des signaux électriques.

Ces propriétés de régularité sont d'ailleurs largement exploitées dans toutes les chaînes de CAO, mais à un niveau plus fin lorsqu'il s'agit, par exemple, d'intégrer des chemins de données : chaque opérateur N bits est obtenu en empilant verticalement N fois la même cellule 1 bit. Le chemin de données est ensuite construit par l'aboutement de ces opérateurs. Le résultat est une structure topologique à deux dimensions dont chaque ligne correspond à un 1 bit. Les dessins de masques ainsi produits sont très compacts. On peut également considérer les générateurs de RAM, de ROM, de PLA, de multiplieurs, etc, où l'on observe la préservation d'une très forte régularité.

Dans tous les cas, l'usage de ces outils est limité à la génération d'un type d'architecture prédéfinie : un chemin de données, une mémoire, un PLA.

Cela signifie qu'une chaîne de CAO doit incorporer plusieurs générateurs, chaque générateur répondant à un besoin particulier. Cette approche n'est évidemment pas viable pour l'intégration des architectures systoliques en général. On n'imagine mal une bibliothèque contenant toutes les applications possibles des réseaux systoliques !

Aussi, l'approche considérée pour pallier ces limitations consiste à adopter une stratégie qui s'apparente à celle des compilateurs de compilateurs. A travers un langage de description on indique ce que doit produire le générateur, tant d'un point de vue fonctionnel que topologique [103]. Les primitives de base sont des portes et des éléments de mémorisation que l'on combine et assemble en structures régulières. Cette approche peut être intéressante pour des cellules systoliques peu complexes qui se résument, par exemple, à un seul chemin de données. L'outil de synthèse peut facilement produire une description compacte des cellules et indiquer comment elles s'aboutent.

Mais dès lors que la cellule devient complexe ou que l'agencement topologique se heurte à des contraintes physiques (par exemple implanter un réseau linéaire sur une grille bi-dimensionnelle), ce type d'outil ne suffit plus. Il faut prévoir des mécanismes de routage plus sophistiqués que le simple aboutement de cellules. Le logiciel MadMacs [81] [34] est un bon exemple de ce que peut être un générateur de générateurs de réseaux réguliers : dans un premier temps, le concepteur définit interactivement le placement et le routage par le biais de commandes basées sur la notion de déplacements relatifs (aller à droite, se positionner sur le n-ième connecteur, etc). Cette suite de commandes est mémorisée dans un *script* que l'on peut ensuite modifier ou paramétrer, et qui constitue le générateur à proprement parler.

Ce type d'outil ne se retrouve pas dans les systèmes de CAO traditionnels. Ce n'est d'ailleurs pas étonnant puisqu'en amont la notion de régularité n'est pas utilisée ! Aujourd'hui, implanter un réseau régulier de processeurs sur silicium avec les outils proposés en standard est possible, mais relativement frustrant. L'expérience du circuit API256, circuit réalisé semi-automatiquement et qui contient seulement 4 processeurs connectés linéairement, a mis en évidence, et de manière criante, les lacunes dans ce domaine. Par exemple, avec le système de CAO que nous avons utilisé (mais cela aurait sans doute été identique avec tout autre système), il n'y a pas de moyens simples, immédiats et automatiques pour guider la synthèse logique d'un processeur en vue de son intégration dans un réseau régulier. On n'y arrive cependant en jouant sur diverses contraintes et en utilisant le logiciel de manière détournée. Mais cet exercice exige une maîtrise complète de l'outil de synthèse et demande

un temps inutilement long.

Ainsi, au niveau physique, les outils de CAO VLSI exploitent seulement un type de régularité : celui qui est prévisible dans les structures prédéfinies comme les mémoires, les PLA, les chemins de données, etc. Dès lors que l'on s'écarte de ces entités, la régularité doit être gérée manuellement.

2.5 Situation des travaux

Les paragraphes précédents ont montré trois aspects sur lesquels la régularité peut être mise à profit pour concevoir des circuits systoliques : la synthèse d'architecture, la simulation et le placement/routage.

Mes travaux de recherche se situent en aval de la synthèse architecturale. Les résultats de synthèse constituent, en fait, la matière première sur laquelle s'appuient mes activités, que ces résultats aient été produits automatiquement ou obtenus manuellement. A ce stade, la "systolisation" est supposée résolue, c'est-à-dire que l'architecture est connue, tout comme la fonctionnalité des cellules.

Mais comme nous avons essayé de le montrer, cette étape de synthèse ne suffit pas pour garantir un silicium régulier. Encore faut-il que les outils des étapes suivantes soient capables de traiter intelligemment cette information ! C'est sur ces différents aspects de la conception d'architectures parallèles intégrées qu'ont porté mes recherches et ce, en suivant un ordre chronologique quelque peu bouleversé par rapport à la manière dont on conçoit un circuit VLSI.

En effet, mes premières recherches se sont concentrées sur l'aspect purement topologique (placement/routage) en développant une technique d'intégration basée sur la notion de noyau. L'idée principale consiste à faire porter l'étude d'une structure hautement régulière sur une version réduite mais représentative du circuit. Cette méthode a été validée par le circuit API69, un réseau systolique bi-dimensionnel dédié à la correction orthographique.

Puis, en collaboration avec Frédéric Raimbault, dont j'ai co-encadré la thèse, mes travaux se sont tournés vers l'autre extrémité de la chaîne de conception (en ce qui me concerne, après la synthèse architecturale), la simulation systolique à un niveau algorithmique : comment décrire simplement, dans un langage de programmation, un algorithme systolique, et comment le simuler efficacement sur une structure parallèle ? Ces travaux ont débouché sur l'élaboration d'un langage, le langage C-stolic, et sur un environnment

de programmation permettant l'exécution de programmes systoliques sur différentes machines parallèles.

Enfin, mes derniers travaux ont tenté d'établir la liaison entre ces deux maillons en proposant une méthodologie de travail pour dériver (presque) automatiquement une structure matérielle régulière à partir d'une description C-stolic. Le support d'étude a été la machine SAMBA, un réseau systolique linéaire dédié à la comparaison des séquences biologiques. C'est un projet qui, entre autre, a permis le déroulement de deux thèses – que j'ai également co-encadré –, celle de M. Hadji [44] sur les aspects implémentation physique du processeur de SAMBA, et celle de P. Guerdoux-Jamet [42] sur les aspects applications.

A travers ces trois projets tournés vers la conception d'architectures systoliques intégrées, plusieurs autres centres d'intérêt n'ont jamais cessé d'être présents, même s'ils n'apparaissent pas clairement ici. J'en citerai trois :

- le premier est la volonté de concevoir des machines parallèles équilibrées et complètes, c'est-à-dire des systèmes qui puissent exploiter efficacement les réseaux systoliques intégrés. Ainsi le circuit API69 ou la machine SAMBA, par exemple, intègrent des dispositifs d'entrées/sorties en rapport avec le volume d'information traité par les réseaux. L'expérience montre que cette partie est aussi difficile à concevoir (sinon plus) que le réseau en lui-même. Les problèmes d'interfaçage, même en présence de structures synchrones comme les réseaux systoliques, sont extrêmement délicats à traiter. C'est un problème de co-design difficile sur lequel, avec R. McConnell, étudiant en thèse, [70], nous avons essayé d'apporter quelques réponses [73].

  Cette volonté de prendre en compte un système complet se retrouve également dans le langage C-stolic où la description des traitements parallèles (exécutés sur le réseau) et la description des traitements séquentiels (exécutés à l'extérieur du réseau) reçoivent la même attention.

- le second est l'étude parallèle d'une technologie alternative au VLSI : la logique reconfigurable. Depuis sa naissance, cette technologie a énormément progressé : la capacité des composants FPGA d'aujourd'hui permet d'envisager d'autres usages que ceux qui leur étaient initialement réservés.

  La conception d'un correcteur orthographique sur PeRLe-0 [102], la compilation de GAMMA [4] [101] – qui a donné lieu à la thèse de

M. Vieillot [100] et que j'ai encadré – ou la conception d'un filtre systolique pour la comparaison de séquences d'ADN sur PeRLe-1 [37] [39], l'accélération des communications sur la machine ArMen [58] ou la réalisation de l'interface de SAMBA, sont autant d'expériences qui m'ont permis d'apprécier les possibilités de cette approche.

- le troisième est l'investissement permanent dans un domaine d'applications, en l'occurrence, ici, le traitement des séquences (*string processing*). On ne peut pas concevoir une architecture performante sans connaissance approfondie de l'application pour laquelle elle est dédiée. Cela impose, d'une part, un apprentissage initial non négligeable et, d'autre part, une coopération suivie avec des chercheurs d'autres disciplines. La machine SAMBA ne serait sans doute pas ce qu'elle est sans l'étroite collaboration avec mes collègues biologistes !

3

Conception du circuit Api69

Ce chapitre illustre mes activités de recherche sur l'exploitation des propriétés topologiques des structures régulières. Le circuit API69 en est le support [56]. La mise en œuvre de cet immense tableau de cellules *full custom* s'est faite sur une version réduite (un noyau) avant de produire automatiquement le dessin de masque et son envoi en fabrication.

Le but de ce chapitre n'est pas de décrire en détail l'architecture du circuit, mais de se focaliser sur une partie de l'étude, celle ayant trait à la méthodologie de conception imaginée pour pallier certaines limites des outils de CAO VLSI.

Nous situons d'abord le contexte de l'étude et évoquons les difficultés rencontrées pour implémenter une structure régulière *full custom* avec les outils de CAO VLSI conventionnels. Nous brossons ensuite un rapide portrait du circuit API69 en insistant sur les aspects topologiques. La méthodologie de conception est alors présentée ainsi que les incidences sur la conception du circuit. Nous concluons par une analyse de cette étude.

3.1 Contexte de l'étude

Le circuit API69 fait suite à la réalisation d'un prototype d'une machine systolique bi-dimensionnelle (SuperMics [53] [96]) pour la correction des adresses postales après lecture optique. Ce prototype s'inspire très largement de l'architecture MicMacs [31] : il réutilise les briques de bases (circuit API15C [28] [29] [27]) et le module de gestion des entrées/sorties. Seule la structure du réseau est changée : un réseau bi-dimensionnel tronqué de 28 processeurs remplace un réseau linéaire de 15 processeurs (MicMacs).

API69 est une structure hautement régulière parfaitement adapté à une implémentation à la Mead & Conway, c'est-à-dire une organisation physique du

plan de masse sous forme de tableau de cellules [74]. L'avantage de cette disposition réside dans l'inexistance de routage, les connexions étant réalisées par aboutement des cellules, ce qui entraîne une haute densité d'intégration. Le revers de la médaille est que les cellules étant optimisées pour ce circuit particulier, elles sont uniques et non disponibles en bibliothèque : il faut les créer de toutes pièces.

Cette approche *full custom* est évidemment moins directe que l'usage de bibliothèques. Elle demande plusieurs opérations supplémentaires dont la création des cellules et leur vérification. Cette dernière opération établit la correspondance entre le dessin de masques et le schéma électrique. Elle intervient à plusieurs niveaux : pendant l'élaboration d'une cellule de base (additionneur 1 bit, par exemple), puis lorsque ces cellules sont assemblées pour former des opérateurs (additionneur/accumulateur 8 bits, par exemple) et, enfin, au niveau de l'assemblage complet du circuit.

Plus précisément, cette opération, appelée extraction/comparaison, détecte, à partir du dessin des diverses couches technologiques, la présence de transistors, puis établit une liste d'interconnexions. Cette liste est confrontée à une autre liste, issue du schéma électrique, pour s'assurer de l'équivalence. En pratique, si les outils de CAO VLSI gèrent très bien l'extraction/comparaison de quelques milliers de transistors, il n'en va pas de même lorsque les listes contiennent plusieurs centaines de milliers de transistors (300 000 pour API69). Les temps de calcul sont impressionnants et l'espace mémoire requis considérable.

Aujourd'hui (5-6 ans plus tard), grâce à l'évolution technologique, les outils et les machines savent, sans doute, manipuler un volume de données équivalent au circuit API69, même si les temps de calcul sont encore longs. Ce n'était pas le cas lors de notre étude (début des années 1990) et ce n'est probablement toujours pas le cas aujourd'hui pour des circuits d'un ou deux millions de transistors, ce qui correspond, grosso modo, au nombre de transistors que l'on pourrait actuellement intégrer sur une surface de silicium identique à celle d'API69.

La méthodologie que nous avons imaginée – pour éviter l'opération d'extraction/comparaison – reste donc valide pour qui veut mettre en œuvre des structures régulières *full custom*. Elle repose sur la notion de noyau et consiste à faire porter les diverses étapes de la conception sur une version réduite, mais représentative du circuit. Avant de décrire cette approche, nous présentons d'abord, et de manière succincte, quelques caractéristiques du circuit API69.

3.2 Le circuit API69

La principale application du circuit API69 est la correction orthographique. A partir d'un mot erroné on calcule une distance avec tous les mots d'un dictionnaire afin de déterminer ceux qui sont les plus proches. Les mots sélectionnés sont alors proposés comme correction. Il faut noter que les fautes de grammaires ne sont pas incluses dans ce type de correction.

En français, la taille moyenne des mots est de 7-8 caractères ; au delà de 15 caractères, les mots ne sont pas assez nombreux pour nécessiter un dispositif matériel de correction. Une structure bi-dimensionnelle de 15×15 processeurs est donc suffisante. Elle peut même être réduite à quelques diagonales sans grandes conséquences sur les résultats. L'architecture du circuit API69 est donc constituée d'un réseau bi-dimensionnel de 15×15 processeurs tronqué à cinq diagonales, soit 69 processeurs.

Les performances du circuit ont été dictées en partie par l'application du tri postal. L'aspect temps réel demande une puissance de calcul équivalente au traitement d'un dictionnaire de plusieurs millions de références par seconde. En se basant sur une fréquence du circuit de 25 MHz et un cycle systolique composé de quelques cycles machines, on atteint aisément cet objectif.

Fonctionnellement, API69 peut être découpé en cinq modules : un réseau de processeurs, une mémoire, un canal d'alimentation, un registre de configuration/test et un module de contrôle.

Le tableau de processeurs comporte 75 processeurs (5 lignes de 15 colonnes). Six processeurs fictifs ont été rajoutés pour préserver la régularité. Ils sont situés sur les bords et ne jouent aucun rôle, sauf celui de propager les données, les commandes et les lignes d'alimentation électrique. Un processeur est composé d'un opérateur spécifique d'addition / minimisation, de quelques registres et de trois ports de communication.

La mémoire contient 15 bancs identiques (un banc par colonne). Elle stocke des données qui paramètrent l'application. Un banc mémoire est accessible par cinq processeurs d'une même colonne.

Le canal d'alimentation répartit les mots du dictionnaire aux extrémités du réseau. Il est simplement composé de registres à décalage qui délivrent, aux bons instants, aux bons endroits et dans le bon ordre les données en provenance du dictionnaire.

Le registre de configuration/test a la double tâche d'initialiser le circuit

(registre des processeurs, bancs mémoire) et de le tester. A partir de ce dispositif, tous les registres des processeurs et toutes les mémoires peuvent être configurés avec des valeurs externes. Réciproquement, l'état de tous les registres et de tous les mots mémoire est visible de l'extérieur.

Le module de contrôle gère l'ensemble. Il reçoit des micro-instructions qu'il décode pour émettre des micro-commandes à à destination de chaque unité.

D'un point de vue mise en œuvre sur silicium, le découpage en cinq modules est conservé. Ces cinq modules sont directement aboutés, sans routage supplémentaire. Chaque module est composé de blocs, eux mêmes composés de cellules. L'ensemble représente l'assemblage de plusieurs milliers de cellules.

3.3 Le noyau

Le circuit API69 est un tableau de plusieurs milliers de cellules *full custom*. Sur une telle structure, l'équivalence entre un schéma électrique et le dessin de masques est impossible à vérifier : l'opération d'extraction/comparaison est trop longue et requiert une configuration mémoire gigantesque. Cette étape est pourtant indispensable pour s'assurer que le dessin de masque reflète exactement le schéma électrique.

L'idée de noyau est de faire porter la validation sur une structure réduite mais représentative du circuit, c'est-à-dire une structure qui possède les mêmes propriétés, à l'échelle de la réduction près. De cette manière, l'opération d'extraction/comparaison s'effectue avec une structure compatible avec les outils de CAO VLSI. La conception du circuit peut alors se résumer en quatre étapes principales :

(1) dessiner et caractériser les cellules de base ;

(2) générer automatiquement un noyau ;

(3) vérifier, simuler, comparer/extraire, etc, le noyau ;

(4) générer automatiquement la version finale du circuit avant envoi en fabrication.

Travailler sur un noyau autorise, en premier lieu, l'opération d'extraction/comparaison, opération quasi impossible sur des structures régulières de grandes

tailles. Mais au delà de cette possibilité, cette approche présente l'avantage d'accélérer globalement le processus de conception. En fait, deux autres opérations coûteuses en temps de calcul bénéficient d'une étude centrée sur un noyau :

- la simulation au niveau transistor : tôt ou tard, il faut simuler finement le comportement du circuit. Cette opération intervient souvent dans la boucle de validation, et passer de quelques heures (API69 complet) à quelques minutes (noyau) autorise une meilleure vérification : le concepteur n'hésite pas à réitérer plusieurs fois le processus de simulation.

- la vérification des règles de dessin : cette étape est obligatoire lors de la conception d'un circuit *full custom*. Elle intervient pendant le dessin de la cellule et à chaque fois qu'un assemblage de cellules intervient. C'est généralement une opération assez longue. Dans ce cas, l'idée de noyau s'apparente au concept de vérification hiérarchique qui s'appuie sur la connaissance de la structure du circuit pour limiter les zones de vérification.

Le noyau d'un circuit est donc un modèle réduit où les propriétés topologiques et fonctionnelles sont préservées. L'opération d'extraction/comparaison devient alors possible tandis que les opérations de simulation fine et de vérification de règles de dessin sont allégées.

3.4 Conception du circuit API69

La méthodologie de conception du circuit API69 s'est faite à partir d'un noyau dont les principales caractéristiques sont un chemin de données sur 4 bits, une mémoire réduite à 3 mots, 6 colonnes de processeurs et 3 diagonales.

Sur cette structure, la mise en œuvre des algorithmes de correction est possible et tous les cas de figure permis. D'un point de vue strictement topologique, le noyau aurait pu être plus petit dans la mesure où 6 colonnes de processeurs ne sont pas indispensables pour vérifier toutes les combinaisons d'aboutement des cellules.

La figure 3.1 résume les différentes étapes de la conception. Les cellules sont d'abord dessinées et caractérisées avec un outil de CAO VLSI (dans notre cas SOLO 2000 de CADENCE). Chaque cellule est représentée par

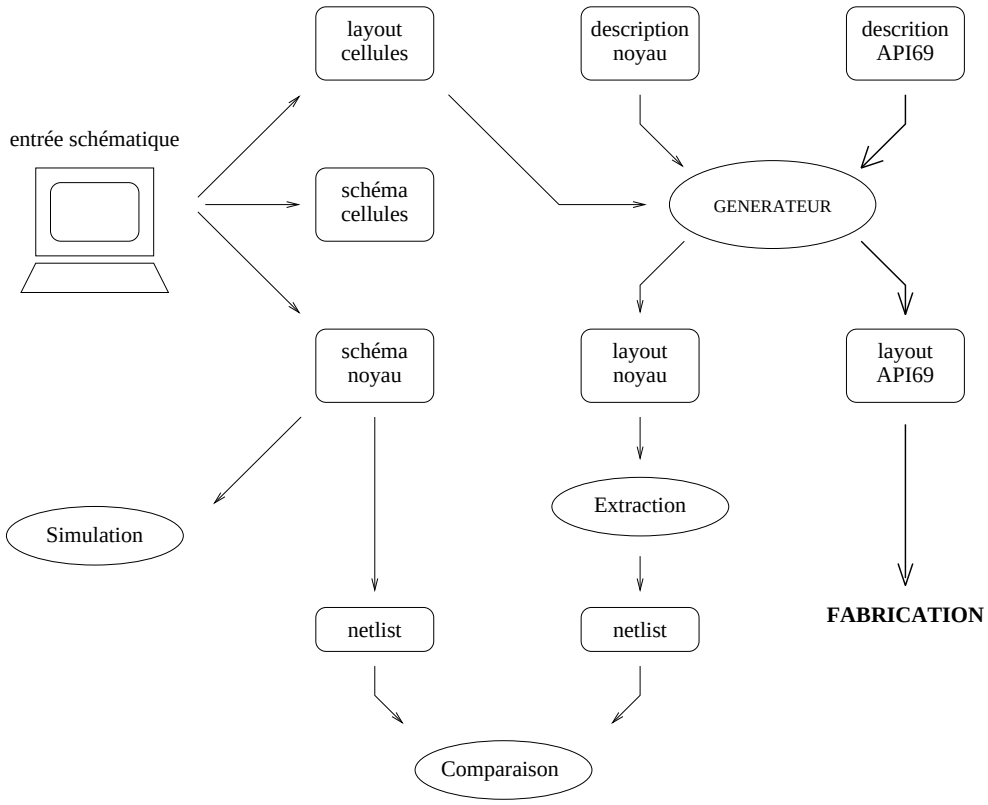

Figure 3.1 : *Méthodologie de conception du circuit API69 : une bibliothèque de cellules (layout et schéma) est d'abord constituée. Un noyau est ensuite établi. Il se compose d'une spécification (schéma électrique) et d'un dessin de masque (produit automatiquement). L'étude complète – simulation, extraction, comparaison, vérification des règles de dessin – est alors réalisée sur le noyau. Lorsque le noyau est au point, le circuit final est généré et envoyé en fabrication sans vérification supplémentaire.*

un dessin de masques et un schéma électrique équivalent. Le résultat de cette première étape est une bibliothèque de cellules prévues pour s'abouter correctement, sans dispositif de routage supplémentaire. Cela suppose, au départ, l'établissement de contraintes topologiques qui doivent être vérifiées tout au long du processus de création des cellules.

Les schémas électriques de ces cellules sont ensuite assemblés pour produire le schéma électrique du noyau. Cet assemblage a été fait manuellement : dans l'environnement de CAO utilisé il n'existait pas de moyen automatique pour générer un schéma électrique à partir d'une description régulière de schémas de base. Des simulations répétées vérifient et corrigent ce schéma.

Le dessin de masques du noyau est ensuite généré automatiquement à partir de sa description. Le générateur est un programme (écrit en C) dont les paramètres d'entrées représentent les caractéristiques du circuit, et dont le résultat est un tableau de symboles ; chaque symbole est un lien vers une cellule de la bibliothèque. A partir de ce tableau, le logiciel de CAO construit le dessin de masques en substituant les cellules aux symboles.

L'étape suivante, l'opération d'extraction/comparaison sur le noyau, valide la cohérence entre le dessin de masques et le schéma électrique. Cette opération met en évidence les erreurs topologiques, comme par exemple, de mauvaises connexions entre cellules. Dans de telles situations, deux cellules, prises indépendamment, sont correctes mais leur assemblage physique ne correspond plus à la connexion électrique.

L'étape finale produit automatiquement le circuit complet. Aucune intervention manuelle n'a lieu, à l'exception des connexions du cœur du circuit aux pattes d'entrées/sorties.

3.5 Analyse

Le premier commentaire qui peut être fait est que le circuit résultant de cette méthodologie de conception a fonctionné du premier coup. Pour des raisons budgétaires, la surface de 1 cm^2 initialement prévue pour supporter un réseau de 69 processeurs a été réduite à 70 mm^2, ce qui (à l'époque!) n'a permis d'intégrer qu'un réseau de 34 processeurs (réseau de 12 $\times$12 à 3 diagonales). Si cette réduction limite la portée applicative du circuit, elle ne change en rien la démarche poursuivie pour le concevoir.

En fait, cette modification de dernière minute nous a conforté dans notre ap-

proche : générer automatiquement cette configuration intermédiaire relève de la même complexité que la génération de la configuration complète. Cela n'a été ni plus ni moins rapide, et ni plus ni moins difficile. Si, d'emblée, l'étude avait porté sur la structure complète, il aurait fallu reprendre l'ensemble des étapes de conception et vérification (simulation, extraction, comparaison, vérification des règles de dessin, etc) pour aboutir à cette nouvelle version. Cette contrariété budgétaire a involontairement mis en évidence le gain en temps de conception lorsque de telles contraintes surgissent à l'improviste.

Nous avons déjà souligné les points sur lesquels le noyau apporte une contribution essentielle (l'extraction/comparaison) et sur lesquels il permet un gain de temps appréciable (simulation niveau *switch*, vérification de règles de dessin). Il est également apparu que travailler sur une petite structure induisait un confort directement lié à l'usage des outils graphiques : quel concepteur n'a pas maudit, pendant une phase de mise au point, les 3 minutes que prennent la sur-coloration d'un nœud électrique appartenant à un circuit dont la taille est à la limite de ce que peut supporter le système ? Dans notre cas, le noyau représentait un volume de données suffisamment faible pour qu'il soit capable d'inter-réagir quasi instantanément aux sollicitations de l'opérateur.

Si le circuit qui a été fabriqué s'est révélé fonctionnellement correct, ses caractéristiques électriques sont, par contre, en deçà de ce qui avait été prévu. La fréquence d'horloge est de 10 MHz au lieu d'une fréquence estimée de 25 MHz. Cette différence provient essentiellement d'un mauvais dimensionnement d'un composant : l'amplificateur du signal d'horloge. La taille de cet amplificateur était suffisante pour que le noyau fonctionne à 25 MHz, mais insuffisante dans le cas du circuit final. On touche, ici, un des points faibles de notre approche, à savoir l'estimation des performances électriques du circuit à partir du noyau.

Une estimation précise est généralement possible après rétro-annotation des *netlist* par la valeur des capacités associées aux fils d'interconnexion. Or, le dessin de masque du noyau ne peut délivrer directement cette information. Il faudrait donc compléter cette étude par un outil qui, à partir des caractéristiques électriques du noyau, projette celles du circuit final. On peut imaginer que ces caractéristiques puissent être prédites en s'inspirant, par exemple, des techniques mises en œuvre dans les outils de CAO VLSI actuels pour les générateurs de chemin de données ou les générateurs de mémoire.

Un autre point d'interrogation de cette approche est le choix du noyau. Quelle doit être la taille minimale du noyau pour s'assurer de sa représentativité vis à vis du circuit final ? D'un point de vue topologique, les règles

semblent simples. Par exemple, vérifier qu'un tableau de $n \times n$ cellules identiques (orientées dans le même sens) s'aboutent correctement consiste à vérifier un sous tableau de taille 2×2 : toutes les combinaisons d'aboutement sont reproduites dans cette mini structure. Ainsi, à partir de la description initiale et de règles de réduction doit-on pouvoir produire automatiquement un noyau qui, topologiquement, reproduit toutes les combinaisons d'aboutement des cellules.

Par contre, s'assurer que le noyau possède toutes les propriétés fonctionnelles du circuit final semble d'un niveau de complexité très supérieur. Pour une mémoire, on imagine facilement que les mécanismes mis en place sont, jusqu'à un certain point, indépendants de sa taille (nombre de mots, taille des mots) ; de même, pour un opérateur arithmétique simple la structure est indépendante du nombre de bits sur lequel il opère. Mais dès lors qu'une structure plus complexe apparaît (l'interaction de plusieurs éléments, par exemple), il devient difficile d'extraire la régularité "fonctionnelle". Sur le circuit API69, par exemple, il n'a pas été immédiat de mettre en évidence que les processus d'initialisation d'une matrice tronquée à cinq diagonales (le circuit final) était équivalent à celui d'une matrice tronquée à trois diagonales (le noyau).

Le noyau d'API69 a été déterminé manuellement à partir du savoir-faire et d'une réflexion ponctuelle sur ce cas particulier. A travers cette expérience, l'élaboration automatique d'un noyau topologiquement correct nous semble possible (aucune recherche sérieuse n'a cependant été menée pour l'affirmer). Ce noyau, restreint aux seules contraintes topologiques, serait beaucoup plus petit que celui incluant l'ensemble des contraintes et pourrait être utilisé, par exemple, pour la vérification des règles de dessin, opération coûteuse et indépendante de la fonctionnalité du circuit.

En revanche, la construction automatique d'un noyau fonctionnellement représentatitf du circuit final nous semble beaucoup plus complexe. C'est sans doute à un niveau de description plus élevé qu'il faut rechercher des solutions. Un langage comme ALPHA, par exemple, pourrait fournir une description paramétrée du circuit pour étudier l'équivalence entre le circuit à réaliser et une version réduite en guise de noyau.

4
Le langage C-stolic

Bien que le langage C-stolic ait été conçu à l'origine pour la programmation de machines systoliques, ce chapitre présente le langage sous l'angle de la conception d'architectures systoliques intégrées. Deux aspects sont abordés : la spécification d'algorithmes systoliques en vue de leur intégration et leur simulation sur machines parallèles. Ce chapitre indique également quelle suite à donner à C-stolic pour qu'il s'oriente vers un outil d'aide à la conception d'architectures systoliques intégrées.

4.1 Historique

Le besoin d'un outil pour la programmation de réseaux systoliques est né aussitôt après la mise au point de la machine MicMacs [54] réalisée pour prototyper des applications orientées vers le traitement des chaînes de caractères. L'architecture de cette machine – un réseau linéaire de 18 processeurs programmables – est basée sur le modèle OSIMD [60] [48], modèle dont la principale caractéristique est de gérer les entrées/sorties en recouvrement avec le calcul. Concrètement cela signifie que deux processus évoluent de manière indépendante : le premier prépare les données tandis que l'autre les traite.

L'écriture de deux programmes pour spécifier le comportement des deux processus s'est avérée très rapidement être une tâche fastidieuse dès lors que les synchronisations et les communications inter-processus devaient être décrites manuellement. De plus, MicMacs étant un accélérateur connecté à une machine hôte, un troisième programme devait spécifier très précisément les échanges entre l'hôte et l'accélérateur.

La motivation initiale de C-stolic a donc été de simplifier la tâche du programmeur en réduisant au maximum la gestion des synchronisations entre

les trois processus (hôte, E/S, calcul) qui, dans les faits, constituait un véritable frein à l'usage de ce type de machine. L'autre objectif, beaucoup plus immédiat, était de proposer un langage de plus haut niveau que celui dont on disposait (un assembleur !) pour programmer efficacement la machine MicMacs.

Une réflexion importante, qui a débuté au travers des stages de DEA T. Le Sergent [66] et F. Raimbault [86] a permis de mettre en place les premières bases du langages. Ce dernier, au cours de sa thèse, a ensuite étendu son rôle à la simulation d'algorithmes systoliques sur des machines parallèles. La compilation de C-stolic ne produit pas de code exécutable pour une machine donnée mais un code C accepté par les compilateurs natifs de diverses machines parallèles. Ainsi, associé à des bibliothèques dépendantes des machines cibles, le processus de compilation adapte automatiquement le modèle OSIMD au matériel disponible.

Si au départ C-stolic visait essentiellement la programmation d'une machine précise, en l'occurrence MicMacs, c'est aujourd'hui un langage de programmation pour une classe beaucoup plus large de machines parallèles. En fait, le modèle de machine supporté par C-stolic est une représentation abstraite de machine. La mise en œuvre peut être tout autre comme l'a montré F. Raimbault en réalisant, par exemple, une implémentation sur un iPSC/2 dont la structure est relativement éloignée de celle d'une architecture systolique.

Ainsi, de l'objectif initial qui était de développer un langage de programmation pour une machine particulière – la machine MicMacs –, le langage C-stolic a-t-il dérivé vers un outil de simulation. Dans le cadre qui nous intéresse ici, la conception d'architectures systoliques intégrées, la simulation constitue l'étape suivant la phase de spécification. A l'usage, il s'est avéré que C-stolic était également un support intéressant pour spécifier le comportement d'un futur circuit, notamment lorsqu'il s'agissait de décrire le système dans son ensemble : calcul des processeurs et traitement externe au réseau.

Notons enfin que l'objectif initial, un langage de programmation pour une machine, n'a pas été immédiatement abandonné : le projet MOVIE [8] [3] a retenu C-stolic comme langage de programmation. Parallèlement, la version suivante de la machine MicMacs[1] (à base de processeur API-RISC) imaginée par Y. Robin pendant son stage de DEA [94] a été doté d'un compilateur produisant un code optimisé à partir de C-stolic (stage de DEA de d'O. Reichard [91]).

[1] cette version de MicMacs est restée à l'état de projet papier : elle n'a pas donné lieu à la réalisation d'un prototype !

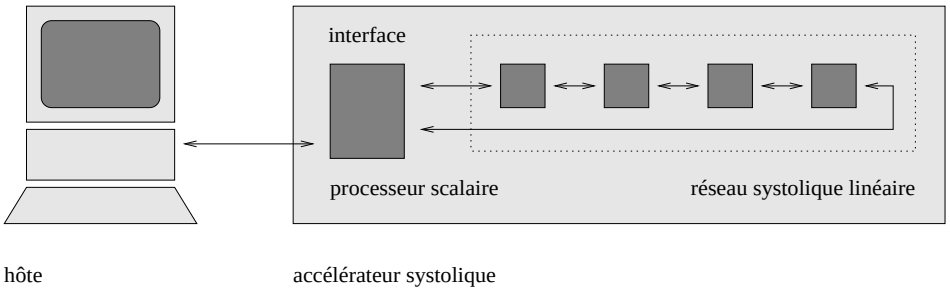

Figure 4.1 : *accélérateur systolique tel que le voit le programmeur : un ordinateur hôte qui communique via une interface programmable (un processeur scalaire) avec le réseau.*

La suite de ce chapitre n'aborde pas cet usage de C-stolic puisqu'il n'est pas orienté vers la conception d'architectures systoliques intégrées. Après une présentation du langage C-stolic, nous montrons l'adéquation de C-stolic à spécifier diverses applications. Nous rappelons ensuite les résultats de la thèse de F. Raimbault sur la simulation parallèle à partir de C-stolic, puis nous terminons par quelques perspectives de recherche.

4.2 Le langage C-stolic

Ce paragraphe introduit le langage C-stolic. Il indique notamment comment, d'un point de vue programmation, les tâches fastidieuses de synchronisation entre les différents processus qui caractérisent le modèle OSIMD sont gérées.

Communication hôte/accélérateur

A l'origine, C-stolic a été imaginé pour programmer un accélérateur systolique, c'est à dire un réseau linéaire de processeurs connecté à un processeur hôte. L'image que se fait le programmeur d'un tel système est celle représentée par la figure 4.1 : un ordinateur hôte qui communique via une interface programmable (un processeur scalaire) avec le réseau. Globalement, cela conduit à considérer deux processus, celui qui s'exécute sur l'hôte et celui qui s'exécute sur l'accélérateur. Rappelons que C-stolic spécifie uniquement ce qui se passe sur l'accélérateur.

D'un point de vue programmation, le lien entre les deux processus est assuré par appel de procédures. Plus précisément, un programme complet (une

application) est décrit par un programme C qui communique avec l'accélérateur par appel de procédures écrites en C-stolic. L'échange d'informations est spécifié par les paramètres des procédures.

Ce mécanisme de communication par appel de procédure élimine la gestion explicite des synchronisations entre l'hôte et l'accélérateur. Le protocole matériel est caché au programmeur ; il est spécifié dans une bibliothèque d'entrées/sorties et dépend de la machine parallèle utilisée.

Ainsi, un des premiers objectifs de C-stolic qui visait une gestion plus facile des synchronisations est atteint : les communications entre l'hôte et l'accélérateur n'apparaissent pas explicitement, mais par le truchement d'appels de procédure. De plus, cette technique, associée à des bibliothèques d'entrées/sorties adaptées, ne restreint pas l'usage de C-stolic à une machine spécifique. Au contraire, les mises en œuvres réalisées sur diverses machines parallèles prouvent qu'il s'agit d'un mécanisme simple et général.

Le modèle de programmation C-stolic

Le paragraphe précédent a exposé le mécanisme de synchronisation entre l'hôte et l'accélérateur. Nous présentons maintenant comment C-stolic résout la synchronisation entre le processeur scalaire (l'interface) et le réseau.

En premier lieu, le programmeur doit considérer une architecture virtuelle ; elle est représentée par la figure 4.2. Elle se compose d'un réseau linéaire de `NBCELL` processeurs et d'un processeur scalaire. Ces deux entités sont pilotées par un séquenceur unique.

Les processeurs du réseau communiquent seulement de voisin à voisin et seuls les deux processeurs situés aux extrémités possèdent des liens de communication avec le processeur scalaire. Une ligne de diffusion permet cependant d'envoyer simultanément une donnée du processeur scalaire vers tous les processeurs du réseau.

A partir de cette architecture virtuelle, deux types de variables sont à distinguer : celles situées sur le processeur scalaire et celles réparties dans le réseau. Ces deux classes sont respectivement référencées par *variables scalaires* et par *variables systoliques*. En C-stolic cette distinction se fait lors de la déclaration des variables. Par exemple, la déclaration

```
int K,L;
systolic int s,t;
```

spécifie l'usage de deux variables scalaires `K` et `L` et de deux variables systoliques `s` et `t`. Les variables systoliques `s` et `t` sont présentes sur tous les

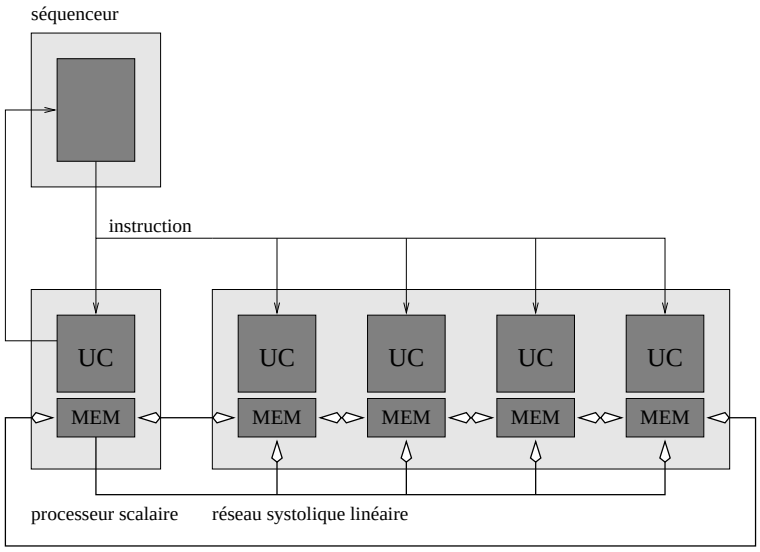

Figure 4.2 : *Le modèle de programmation C-stolic : le programmeur considère un séquenceur unique qui diffuse des instructions au réseau et au processeur scalaire. Seul le processeur scalaire a le pouvoir de modifier le séquencement.*

processeurs. Ce sont, en quelque sorte, des vecteurs de `NBCELL` valeurs. L'opération `s=t+1` effectue `NBCELL` opérations en parallèle, chacune se déroulant sur un processeur du réseau.

L'échange de valeurs entre les processeurs se fait par une opération d'affectation particulière, appelée *affectation systolique*. Cette opération représente une communication de type systolique entre processeurs. Elle est notée par le symbole `=<` pour un transfert droite-gauche et `=>` pour un transfert gauche-droite. Ainsi, en supposant un réseau à trois processeurs dont l'état des variables `s` et `t` est le suivant :

	P1	P2	P3
t	8	7	1
s	2	5	3

l'affectation systolique `s =< t` modifie l'état du réseau comme suit :

	P1	P2	P3
t	8	7	1
s	7	1	?

Après l'affectation systolique, la valeur de la variable `s` du processeur P3 est

indéfinie. Celle-ci peut cependant être spécifiée par une affectation systolique dans laquelle on précise la valeur à émettre vers le réseau. De nouveau, toujours à partir de l'état initial mentionné auparavant, l'affectation systolique `s =< t : -1` conduit aux états :

	P1	P2	P3
t	8	7	1
s	7	1	-1

En réalité, ce complément syntaxique à l'affectation systolique a été introduit pour une raison beaucoup plus fondamentale : il permet l'échange de valeurs entre variables scalaires et variables systoliques.

Une procédure C-stolic manipule deux types d'objet (variable scalaire ou variable systolique) qui inter-réagissent via une opération unique, l'affectation systolique. Cette opération symbolise les échanges de données – donc les synchronisations – entre le processeur scalaire et le réseau. D'un point de vue programmation, la gestion des transferts est alors complètement transparente.

A partir d'une telle description le compilateur C-stolic génère deux programmes : un premier relatif au calcul du processeur scalaire et un second relatif au calcul du réseau. La séparation des tâches s'effectue par l'analyse du type des variables.

Il est important de retenir que ces deux programmes sont générés *automatiquement* à partir d'un source unique et que les difficultés de programmation liées à la spécification et à la synchronisation des processus typiques d'une architectures OSIMD sont donc évacuées. Ainsi, le second objectif qui était de décrire simplement la synchronisation interface/réseau est atteint.

4.3 Spécifier avec C-stolic

Lorsque l'on écrit un programme en C-stolic, on le fait pour une machine virtuelle. L'implémentation de C-stolic sur une station de travail procure alors un outil d'excellente qualité pour tester et mettre au point des programmes. La séquentialité apporte, de plus, un déterminisme indispensable au déboggage.

Bien qu'ayant été défini, à l'origine, pour un type de machine spécifique et pour une gamme d'applications particulières (sur MicMacs et pour le traitement de textes), C-stolic a été testé dans d'autres domaines applicatifs

dont le support final visait une architecture parallèle intégrée. Ces travaux ont eu pour objectif d'évaluer la puissance d'expression du langage et de relever les insuffisances dans le cas d'une orientation de C-stolic vers un outil de spécification. Nous décrivons succinctement quatre domaines sur lesquels C-stolic a été testé et indiquons, si besoin, les extensions imaginées.

Biologie moléculaire : l'ensemble des algorithmes de comparaison de séquences biologiques que nous avons étudiés ont été décrits en C-stolic. Ce domaine étant très proche (sur le plan algorithmique) de la correction orthographique, les structures de machine se ressemblent très fortement. Aussi, C-stolic s'est-il révélé parfaitement adapté à ce domaine.

Compression d'images : à travers MOVIE, une chaîne complète de compression d'images (MPEG) a été décrite. La structure de MOVIE est un réseau de sous-réseaux, où chaque sous-réseau exécute un traitement différent. C-stolic étant inapte à décrire l'organisation du calcul dans son ensemble, G. Le Fol [3], doctorant de l'équipe API, a rajouté, à l'instar de la classe `systolic`, une classe supplémentaire, baptisée `parallel`, pour symboliser un second niveau de hiérarchie.

Capteur d'images intégré : un capteur intégré est un dispositif qui, sur la même puce de silicium, intègre simultanément un capteur optique (matrice de photo-diodes) et une architecture parallèle (un réseau linéaire programmable) pour les traitements de bas niveau. La description de ces calculs en C-stolic serait grandement facilité par une opération qui charge et répartie immédiatement une ligne de la matrice de photo-diodes dans le réseau. L. Le Pape, étudiant en thèse, suggère l'opération de distribution qui répartit un vecteur (de type scalaire) sur les variables systoliques des processeurs [65].

Démodulation numérique : ce projet, étudié en collaboration avec le LEP[2], vise à intégrer, sur une même puce, un réseau de processeurs spécialisés pour supporter la démodulation numérique de trois grands modes de transmission : hertzienne, par satellite et par câble [68]. Chaque mode fait appel à des techniques différentes, mais ont en commun d'enchaîner plusieurs traitements. Comme dans le cas de la compression d'images, C-stolic ne peut décrire globalement l'ensemble des calculs. De plus, et contrairement à MOVIE, les sous-réseaux ont des tailles différentes. Si chacun des traitements (des calculs en arithmétique complexe) a pu être décrit séparément, il reste à imaginer un moyen simple pour les raccorder.

La programmation de divers domaines d'applications révèle l'adéquation de

[2] LEP : Laboratoire d'Electronique de Philips

C-stolic à supporter leur parallélisation sur des structures linéaires, même si quelque fois les limites sont atteintes. Mais la version actuelle de C-stolic n'est pas définitive : c'est un langage qui ne demande qu'à évoluer à partir des concepts généraux qui se dégagent de ces expérimentations.

4.4 Simuler avec C-stolic

Accélérer une simulation au niveau algorithmique est nécessaire, même si, dans la conception d'une architecture systolique intégrée, le temps consacré à cette étape est moindre par rapport à une simulation matérielle. Posséder un simulateur rapide permet cependant des tests plus complets et, par conséquent, augmente la confiance que l'on peut avoir de la spécification.

Pour la simulation de systèmes complexes, tels que la compression d'image (codage 34 Mbits/s entre studio, codage MPEG pour le transfert d'images animées) comme l'a montré Mc Connell dans sa thèse [70], les temps de simulation, même au niveau algorithmique, sont très importants : les simulations portent sur un volume de données conséquent (des séquences d'images) et servent à "régler" certains paramètres. Elles doivent donc être exécutées plusieurs fois. Dans ces conditions, l'accélération des simulations est quasi-obligatoire, surtout pour les parties algorithmiques régulières qui constituent près de 90% du temps de calcul.

La simulation d'algorithmes systoliques en C-stolic sur des machines parallèles est une partie du travail de thèse de F. Raimbault [89]. Dans ce paragraphe, nous décrivons succinctement ce travail et analysons les résultats.

Habituellement, on parallélise un algorithme lorsque la puissance de calcul d'une machine séquentielle est insuffisante. Si la vérification de cette parallélisation s'effectue par simulation, l'opération peut alors être très longue. D'où l'idée de s'appuyer sur le travail du compilateur C-stolic pour accélérer le processus.

F. Raimbault a porté le compilateur C-stolic sur diverses machines parallèles de l'IRISA présentes pendant la durée de sa thèse : l'IPSC/2, le iWarp, et la MasPar. Une implémentation sur ArMen, une machine développée à l'UBO[3], a également été validée. Toutes ces machines ont des structures différentes et démontrent le bien fondé de la technique de portage mise en œuvre.

[3]Université de Bretagne Occidentale

A défaut d'être efficace, les portages réalisés sur l'IPSC/2 et la MasPar ont expérimenté les schémas de compilation pour les architectures MIMD et SIMD. Par contre, le portage sur le iWarp dont l'architecture est proche du modèle C-stolic, a démontré l'efficacité de la parallélisation. Sur certains tests, la supra-linéarité promise par le modèle OSIMD a même été mise en évidence ([89] page 101). Dans ce cas, le facteur d'accélération par rapport à une exécution séquentielle est supérieure au nombre de processeurs utilisés !

Malheureusement, le iWarp n'a pas eu de successeur et les machines possédant une architecture similaire ne sont pas courantes. L'efficacité du iWarp est due à un mode de communication parfaitement adapté au modèle systolique, mode qui permet l'échange d'informations suivant une granularité très fine. Cette caractéristique ne se retrouve pas sur les machines parallèles actuelles, et la circulation de nombreux messages très courts pénalise très fortement les performances du système.

Une alternative, testée partiellement avec la machine ArMen, consiste à établir des connexions privilégiées (point à point) et rapides entre processeurs [58]. Via la couche reconfigurable d'ArMen, des liaisons parallèles directes ont été mises en places, avec un mécanisme de contrôle minimal. Ceci est possible parce que le compilateur C-stolic génère un code qui évite les interblocages et qui maîtrise complètement la gestion des tampons d'entrées/sorties.

Les expérimentations réalisées sur ArMen ont démontré l'intérêt de cette approche : le protocole d'échange (câblé dans les FPGA) est réduit à sa plus simple expression et procure un délai de transmission équivalent à 2 ou 3 cycles machines. Actuellement, l'idée pourrait être étendue en raccordant un réseau de stations de travail via des cartes FPGA[4] étroitement couplées au bus interne des machines et possédant des liens de communications rapides.

4.5 Perspectives

Dans le cadre de la conception d'architectures parallèles intégrées, les évolutions de C-stolic peuvent se situer à différents niveaux :

(1) étendre le langage pour supporter une gamme plus vaste d'applications ;

[4] Un exemple est la carte DVC1 proposée par Virtual Computer : connectée sur le SBUS d'une station de travail SUN, elle émet et reçoit des données sur une ligne série à quelques dizaines de Mbits/seconde (http://www.vcc.com)

(2) trouver d'autres techniques pour accélérer la simulation ;

(3) s'approcher d'une simulation transfert de registres.

Extension du langage

Nous avons déjà signalé les limitations du langage C-stolic pour diverses applications : compression d'images, capteur d'images intégré, démodulation numérique. Ces limitations proviennent du fait que ces réseaux représentent simplement un ou plusieurs sous-systèmes d'un ensemble plus complexe intégré sur un même support de silicium. La conception d'une puce contenant uniquement un réseau régulier de processeurs appartient déjà au passé. Il faut donc doter C-stolic de mécanismes nouveaux qui représentent les actions globales à effectuer sur les réseaux. L'exemple de l'opérateur de distribution permettant d'acquérir simultanément une valeur différente dans tous les processeurs, ou la classe `parallel` permettant la constitution de réseaux de réseaux sont deux illustrations.

L'autre limitation évidente de C-stolic est la restriction à un réseau linéaire. Toutes les implémentations ne se satisfont pas de cette structure. La mise en œuvre d'un filtre de Nyquist [68], par exemple, typique d'un étage de filtrage en traitement du signal, s'effectue plus naturellement sur un double réseau linéaire (plus exactement une matrice $2 \times N$). L'extension de C-stolic dans cette direction ne pose pas véritablement de problème. Une attention toute particulière devra cependant être portée sur la gestion des entrées/sorties. En général, sur des structures bi-dimensionnelle, les problèmes se rencontrent à ce niveau, et non pas au sein du réseau.

Accélération de la simulation

A moins de disposer de machines parallèles permettant la transmission rapide de messages courts (quelques octets), les implémentations du modèle C-stolic manquent d'efficacité. L'inadéquation des machines parallèles aux communications systoliques est principalement responsable de cet état de fait.

L'implémentation de liaisons rapides et dédiées entre processeurs, – comme celles qui ont été réalisées sur ArMen – est une alternative prometteuse qui mériterait d'être testée grandeur nature, sur du matériel récent (station de travail + carte de communication). Cette solution présente l'avantage d'être

immédiate à mettre en œuvre (pas de différence fondamentale par rapport à celle réalisée sur ArMen) et ne nécessite pas de développement matériel particulier (les cartes existent).

Jonction vers le matériel

Ce dernier point est plus prospectif et empiète quelque peu sur le chapitre suivant dans la mesure où, en ayant toujours en tête l'orientation de C-stolic vers un outil d'aide à la conception d'architectures systoliques intégrées, l'idée est de dériver une architecture matérielle à partir de cette spécification.

Dans cet optique, les résultats de la synthèse logique (la transformation du code C-stolic en un ensemble d'opérateurs matériel) pourraient également être utilisés comme point d'entrée pour un accélérateur matériel à base de logique reconfigurable. Le compilateur C-stolic se prête tout à fait à ce schéma : une fois le traitement d'une cellule identifiée, il *"suffit"* de générer un code pour FPGA, et non le code C généré actuellement.

Cette dérivation n'est vraisemblablement possible que pour un code C-stolic écrit en vue de cette transformation (du C-stolic synthétisable ?) et qui représente sans doute un sous-ensemble des possibilités du langage. C'est cette approche qui a été testée dans le cadre de la conception de SAMBA : la spécification C-stolic de la cellule systolique a été écrite de manière à ce que la traduction (manuelle) en un langage de description matérielle (en l'occurrence VHDL) soit la plus naturelle possible.

Pour automatiser cette étape, le langage C-stolic doit être étendu pour prendre en considération certains aspects matériels qui n'apparaissent pas au niveau algorithmique. Par exemple, en C-stolic, on ne spécifie pas la taille des chemins de données ; les calculs sont réalisés dans le format de l'ordinateur, alors qu'une mise en œuvre matérielle demande forcément ce type de précision.

5

Conception de la machine SAMBA

Ce chapitre traite de la conception d'une machine systolique dédiée à la comparaison de séquences biologiques : la machine SAMBA[1]. Ce projet avait le double objectif d'évaluer les capacités du langage C-stolic comme un outil d'aide à la conception d'architectures parallèles intégrées et de proposer une solution viable pour réduire fortement les temps de calcul liés aux traitements des séquences génomiques.

Nous précisons d'abord le domaine d'intervention de SAMBA et présentons l'architecture globale du système. Nous décrivons ensuite les différentes étapes de la conception en soulignant, pour chacune d'elles, l'apport (ou les lacunes !) de C-stolic. Les performances du prototype sont ensuite exposées avant de conclure sur une brève analyse et sur l'avenir de ce projet.

5.1 Domaine d'applications

SAMBA est entièrement dédiée à la comparaison des séquences biologiques [2] [63]. Cette tâche est un des traitements de base de la biologie moléculaire. La dénomination "*comparaison de séquences*" regroupe, en fait, plusieurs opérations qui ont en commun de manipuler de gros volumes de données (les séquences biologiques) et de leur appliquer des traitements informatiques coûteux en calcul.

Le point commun de ces traitements est l'estimation d'une distance entre deux séquences (ou deux sous-séquences) afin de juger de leur *ressemblance*. La distance est déterminée sur la base du meilleur alignement que l'on peut

[1] SAMBA : *Systolic Accelerator for Molecular Biological Applications*

faire entre ces deux séquences. Il existe principalement deux méthodes d'alignement : la première détermine une distance entre deux séquences (ou sous-séquences) de même taille (pas d'insertion/omission de caractères). Les avantages résident dans la possibilité d'inclure des heuristiques pour accélérer le calcul et dans le modèle mathématique sous-jacent pour estimer la validité des résultats. L'inconvénient majeur est que ce mode de calcul est inadapté lorsque l'on doit évaluer une distance entre deux séquences (ou morceau de séquences) de tailles différentes.

La seconde méthode pallie les inconvénients de la première au prix d'une complexité algorithmique accrue. Les méthodes mises en jeu font essentiellement appel à des techniques de programmation dynamique qui impliquent des volumes de calculs importants et conduisent, par conséquent, à des temps de calculs très longs. Aussi, ces méthodes sont elles très peu employées aujourd'hui lorsqu'il s'agit, par exemple, d'explorer des banques de séquences volumineuses.

La machine SAMBA a comme objectif principal de proposer une solution pour réduire fortement les temps de calcul relatifs aux traitements appartenant à la deuxième famille[2], c'est à dire aux traitements qui sont en général laissés de côté parce que trop onéreux en calculs.

5.2 Architecture

Les algorithmes considérés (programmation dynamique) sont des algorithmes *réguliers* qui s'implémentent bien sur des structures systoliques linéaires. Aussi, le cœur de SAMBA est-il composé d'un réseau linéaire de processeurs dédiés.

L'architecture globale de SAMBA comprend quatre éléments : une station de travail, un disque pour stocker les banques de séquences, une interface et le réseau de processeurs VLSI. La figure 5.1 décrit l'interconnexion de ces éléments.

La station de travail (DEC 5000/240) est un modèle standard équipé d'un processeur RISC MIPS R3400 cadencé à 40 MHz. Le choix de cet équipement a été déterminé par la disponibilité de la carte PeRLe-1 (l'interface reconfigurable) qui ne peut se connecter qu'à ce type de machine. Tout autre matériel convient et n'est pas déterminant pour les performances du système. Son rôle

[2]cela n'exclue pas l'accélération des algorithmes appartenant à la première famille

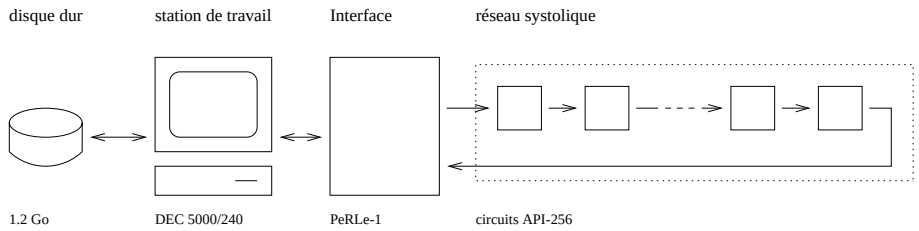

Figure 5.1 : *Synoptique de SAMBA. Le prototype comprend quatre éléments : une station de travail DEC 5000/240, un disque local où sont stockées les banques de séquences, une interface reconfigurable – la carte PeRLe-1 conçue par DEC-PRL – et un réseau de 128 processeurs VLSI réalisés en interne.*

est de supporter l'application principale et de faire appel au réseau lorsque des comparaisons intensives de séquences biologiques sont requises.

L'interface est une carte à base de composants reconfigurables (FPGA). Il s'agit de la carte PeRLe-1 [5] [102] développée par l'équipe de Jean Vuillemin à DEC-PRL. Cette carte concrétise le concept de PAM (Programmable Active Memory) introduit par les concepteurs : l'objectif d'une PAM est d'implémenter une machine virtuelle qui peut être reconfigurée dynamiquement en une infinité d'architectures spécifiques. Après la phase de configuration, la PAM se comporte comme un ASIC. Son rôle est de piloter efficacement le réseau en lui procurant des données au rythme imposé par la cadence systolique. Elle assure également le partitionnement des calculs.

Le réseau systolique est composé de 32 circuits VLSI (baptisés API256) incluant chacun 4 processeurs (soit 128 processeurs). Les circuits ont été réalisés en interne et fondus par la filière universitaire du CMP[3]. La fréquence de fonctionnement est de 10 MHz ; à cette vitesse, la puissance de calcul par puce est de 400 MOPS[4], ce qui confère au réseau une puissance crête de 12,8 Giga opérations/seconde !

5.3 Méthodologie de conception

La conception du circuit API256 s'est appuyée sur le langage C-stolic. Rappelons que l'objectif était d'évaluer les possibilités de cet outil en matière de

[3] CMP:Circuit Multi Projets

[4] MOPS : Millions d'OPérations/Seconde ; une opération équivaut à une addition sur 12 bits

conception d'architectures parallèles intégrées. Aussi, la méthodologie que nous avons suivie mérite quelques explications. Elle se résume à la suite d'étapes suivantes :

(1) spécification en C-stolic ;
(2) simulation en C-stolic ;
(3) spécification en VHDL ;
(4) simulation en VHDL ;
(5) comparaison C-stolic/VHDL ;
(6) synthèse du code VHDL ;
(7) test.

Il peu paraître surprenant que SAMBA ait dû être spécifié et simulé à la fois en C-stolic et en VHDL. La raison principale est simplement dû au fait que, ne disposant pas d'outil de transcription de C-stolic vers un langage compréhensible par un outil de synthèse logique, cette description a été faite manuellement, et qu'il a fallu vérifier, par simulation, que les deux spécifications étaient équivalentes. En fait, les étapes 3, 4 et 5 sont un substitut à une étape de traduction automatique de C-stolic en VHDL.

Les paragraphes suivants détaillent chacune de ces étapes.

Spécification C-stolic

L'algorithme qui a été câblé s'inspire des travaux de Smith et Waterman [98] et Gotoh [41]. C'est un algorithme stable et unanimement reconnu dans le domaine de la biologie moléculaire. Cependant, pour être utilisé dans la plupart des applications, il doit autoriser "certaines" variations.

Un des premiers buts de notre étude a donc été de cerner l'ampleur de ces variations, tout en sachant que le produit final serait un ASIC non programmable. Plusieurs applications cibles ont alors été étudiées, puis le traitement de base relatif à la comparaison de séquences a été parallélisé et simulé en C-stolic. L'étude comparative des codes C-stolic a ensuite généralisé l'algorithme par l'introduction de paramètres qui, suivant leurs valeurs, déterminent l'algorithme à exécuter. Cette étude nous a permis :

- de déterminer un algorithme paramétré couvrant un large spectre d'applications dans lesquelles la comparaison de séquences intervient de manière intensive ;

- d'être certain que cet algorithme était parallélisable (puisque décrit directement en C-stolic) ;

- d'imaginer (en parallèle) un dispositif interne aux processeurs pour acheminer les résultats vers les extrémités du réseau ;

- de prendre conscience que chaque variation algorithmique demande un contrôle du réseau différent (initialisation, alimentation en données et collecte des résultats).

Ce dernier point est particulièrement important car sa mise en œuvre conditionne grandement les performances globales de la machine [60]. Le contrôle ne peut être définitivement câblé puisqu'il dépend de l'algorithme exécuté par le réseau ; il ne peut être exécuté par un processeur séquentiel, trop lent pour gérer, à chaque top d'horloge, l'émission et la réception simultanées des données. C'est pourquoi, nous avons opté pour une structure à base de FPGA, structure qui permet d'allier les contraintes temporelles à une architecture modulable.

La spécification C-stolic a également déterminé la part des tâches à réaliser sur les différents éléments de la machine. L'architecture d'une application peut se décomposer de la manière suivante : le programme C, qui décrit l'action de la station de travail, appelle une procédure C-stolic lorsque des traitements relatifs à la comparaison de séquences sont requis. Les divers algorithmes de comparaison sont spécifiés en C-stolic, et font appel à une fonction unique pour décrire le comportement du réseau. Cette fonction représente la tâche des processeurs au cours d'un cycle systolique et est identique quel que soit l'algorithme exécuté.

Autrement dit, la programmation de divers algorithmes de comparaison consiste à spécifier différemment : (1) les paramètres à stocker dans le réseau, (2) l'émission des données et (3) la récupération des résultats. La fonction du réseau ne peut être modifiée parce qu'elle représente ce qui sera câblé dans le circuit contrairement au contrôle externe qui est supporté par une technologie FPGA.

Le programme C, la procédure C-stolic et la fonction réseau sont donc trois descriptions clairement identifiées qui représentent respectivement les actions de la station de travail, de l'interface et du réseau. C'est à partir de cette dernière description (la fonction réseau) qu'a été déduit le code VHDL du circuit.

Simulation C-stolic

Contrairement à l'étape précédente qui s'est satisfaite de données réduites pour mettre au point le programme C-stolic, cette étape a eu pour but de valider les choix réalisés sur des données grandeur nature. En effet, la mise au point algorithmique s'accomode mal avec la gestion de gros volumes de données et des temps d'exécution qui en découlent. Aussi, pour tester rapidement le bien fondé d'une modification, par exemple, les vérifications sont réitérées sur un petit jeu de données représentatif. Mais, aussi représentatif soit-il, ce jeu de données ne peut prétendre couvrir la majorité des cas, notamment les cas particuliers qui n'ont pas été imaginés et qui posent souvent problème.

Lorsqu'on réalise un ASIC, il est indispensable de garantir sa fonctionnalité dans tous les cas de figure : après passage en fonderie, une remise en cause n'est plus possible, contrairement à un logiciel qui peut toujours être corrigé ultérieurement. La première étape ayant stabilisé l'algorithme – donc l'architecture du processeur résultant – cette seconde étape le valide par le biais de simulations intensives.

Pour réduire ce temps de simulation, nous avons naturellement opté pour les diverses mises en œuvre de C-stolic sur machines parallèles. En fait, c'est à ce stade, que les limites de cette approche sont apparues : une exécution sur la MasPar ou sur le iWarp n'était pas plus rapide que l'exécution sur les dernières stations de travail de l'époque! Cela s'explique, d'une part, par l'évolution technologique des micro-processeurs (8 iWarp à 20 MHz contre 1 Alpha 21064 à 150MHz) et, d'autre part, par l'inéquation de la MasPar à émuler un réseau linéaire.

Spécification VHDL

La description C-stolic du comportement du réseau s'est faite en visant une transcription en VHDL. Nous avons montré précédemment comment nous avions organisé cette spécification pour mettre en évidence les tâches exécutées sur les divers éléments de la machine SAMBA : le traitement effectué sur le réseau est représenté par une série d'instructions encapsulées dans une fonction que l'on appelle à chaque cycle systolique. Ces instructions manipulent que des variables de type `systolic`.

Pour fixer les idées, nous donnons, ci-après, un extrait de la suite d'instructions C-stolic avec, en regard, l'équivalent VHDL :

```
Code C-stolic              Code VHDL

                           Fv_B <= Fv - B;
                           Eh_B <= Eint - B;
                           Hv_A <= Hv - A;
                           Hh_A <= Hint - A;
tmp5 = Hh-A >? Eh-B;       if signed_vector(Hh_A) > signed_vector(Eh_B)
                             then tmp5 <= Hh_A;
                             else tmp5 <= Eh_B;
                           end if;
F = Hv-A >? Fv-B;          if signed_vector(Hv_A) > signed_vector(Fv_B)
                             then F <= Hv_A;
                             else F <= Fv_B;
                           end if;
tmp1 = tmp5 >? F;          if signed_vector(tmp5) > signed_vector(F)
                             then tmp1 <= tmp5;
                             else tmp1 <= F;
                           end if;
tmp2  = Hd+SUB[S];         tmp2 <= Hd + subs;
tmp3  = delta >? tmp2;     if signed_vector(tmp2) > signed_vector(delta)
                             then tmp3 <= tmp2;
                             else tmp3 <= delta;
                           end if;
```

Cet extrait montre la correspondance immédiate entre les lignes de code C-stolic et VHDL. Cela ne veut pas dire que la transcription est juste (il faut s'assurer, par exemple, que la séquentialité C-stolic soit équivalente à l'encapsulation d'un `process` VHDL) mais que la *distance* qui les sépare n'interdit pas tout espoir d'automatisation.

Simulation VHDL - Comparaison avec C-stolic

La simulation VHDL a eu pour but de vérifier que les descriptions C-stolic et VHDL étaient équivalentes. Les vecteurs de test ont été générés en introduisant dans le code C-stolic une fonction d'impression des variables systoliques représentant les connexions inter-processeurs. Cette fonction, placée en tête de la boucle principale, donne à chaque cycle systolique l'état de toutes les liaisons. Ces données peuvent ensuite être reprises à la fois comme stimuli d'entrée du simulateur VHDL et comme données de vérification.

Cette méthode a ses limites : elle n'assure pas une équivalence totale à cause de la non exhaustivité des situations. On peut seulement prédire que plus les tests sont nombreux, plus la chance de laisser passer une erreur est faible.

Synthèse

Cette étape produit les masques du circuit à partir de la description VHDL. L'outil de CAO utilisé a été le synthétiseur ASICSYN de COMPASS et les outils de placement/routage classiques.

La partie synthèse, proprement dite, est immédiate. Le code VHDL est analysé par ASYCSYN et délivre une *netlist* composée d'éléments de bibliothèque. La synthèse peut être partiellement dirigée en décorant le code VHDL de directives telles que le partitionnement en plusieurs blocs arithmétiques, le choix d'opérateurs optimisés (en surface ou en vitesse), etc.

Pour obtenir le circuit final, nous avons procédé en deux temps :

(1) synthèse, puis placement/routage d'un processeur ;

(2) placement/routage du circuit en considérant les processeurs comme des boîtes noires.

Cette démarche correspond à une préservation de la régularité pour optimiser le dessin des masques. L'idée directrice est que l'optimisation d'une petite surface (le plan de masse d'un processeur) produit de meilleurs résultats, et que la juxtaposition de ces surfaces optimisées conduit à un plan de masse intéressant. A l'usage, pour SAMBA, cette approche s'est avérée pénalisante : le placement/routage immédiat de l'ensemble des blocs primitifs (blocs de chemin de données et blocs mémoire) conduit à une surface de silicium plus petite !

Cette expérience semble donc défavorable à la régularité. En fait, il apparaît que l'implantation d'un réseau de quatre processeurs est une structure "insuffisante" pour que les propriétés de régularité puissent être pleinement exploitées. Dans ce cas, une optimisation globale pour placer/router le circuit est préférable.

Par contre, une étude sur une version suivante du circuit et dans une technologie plus fine [93] a mis en évidence que l'intégration de 16 à 20 processeurs n'était efficace qu'en respectant une certaine hiérarchie. La brique de base n'est pas forcément le processeur, mais un petit groupe de processeurs dont le nombre est difficilement prédictible. Seuls, des essais interactifs avec les outils de CAO permettent de trouver le meilleur compromis entre une mise à plat de tous les éléments et une structuration forte.

Test

Les tests ont été réalisés en plusieurs temps. Tout d'abord, les circuits ont été vérifiés un par un avec un générateur/analyseur logique classique. Les vecteurs de test présentés aux circuits ont été produits à partir de C-stolic et les résultats confrontés aux résultats de simulation des programmes C-stolic. A l'issue de cette étape, un prototype de 128 processeurs a pu être réalisé.

S'est ensuite posé le problème du test complet de la machine SAMBA. Il fallait vérifier à la fois que l'architecture implantée sur PeRLe-1 était correcte et que le réseau ne recelait pas de défauts de fabrication (mauvais contact, soudure imparfaite, piste coupée, etc). Nous avons procédé en deux temps : nous nous sommes d'abord assurés que le fonctionnement du réseau VLSI était correct, puis nous avons testé l'ensemble.

Le test du réseau s'est effectué de manière simple, via le simulateur C-stolic. La fonction systolique décrivant le fonctionnement des processeurs a été remplacée par le réseau physique, l'architecture implantée sur PeRLe-1 étant alors réduite à des liens vers le réseau pour acheminer les données et récupérer les résultats. Toutes les simulations algorithmiques précédentes ont été refaites, mais dans un laps de temps beaucoup plus court.

Par contre, le test de l'ensemble a été beaucoup plus délicat. La spécification C-stolic décrivant l'alimentation du réseau, la collecte des résultats ou le partitionnement du calcul avait été faite sans se soucier de l'implémentation matérielle, ni des ressources disponibles sur la carte PeRLe-1. Aussi, la correspondance entre la spécification et la mise en œuvre effective ne s'est pas avérée immédiate et n'a pas aboutie à la définition d'une méthodologie particulière.

5.4 Performances

On ne peut conclure ce chapitre sans donner un aperçu des performances de la machine SAMBA. Celles-ci témoignent, en effet, de l'efficacité des architectures systoliques. Elles montrent également qu'en un temps de conception relativement court (estimé à un homme/année [38]) et avec des choix technologiques très raisonnables, les solutions proposées sont tout à fait viables.

Le prototype fonctionne depuis la fin 1995 et, depuis cette date, diverses expérimentations ont été menées. Mais les mesures des performances ne peu-

vent être qu'appréciées que par rapport à d'autres systèmes, et le choix des concurrents ou des critères de comparaison est toujours délicat : plusieurs facteurs comme le prix, la technologie, la spécificité interviennent et ne font jamais l'unanimité. Aussi, SAMBA étant un accélérateur matériel, c'est-à-dire un dispositif que l'on connecte à un ordinateur standard, nous apprécions principalement les performances de SAMBA sur cet unique critère.

Cela ne veut pas dire que la confrontation avec d'autres systèmes n'est pas intéressante, mais elle est généralement biaisée par les modes d'évaluation (presque toujours) réalisés dans les conditions d'utilisation optimales (typiquement, pour une machine systolique, on se borne à indiquer les performances crêtes du réseau !). Nous pouvons donc simplement situer SAMBA par rapport à d'autres approches, mais difficilement quantifier leurs performances relatives.

A notre connaissance, le projet le plus proche de SAMBA est la machine BISP [17] qui intègre également un réseau de processeurs VLSI dédiés. Les performances (crêtes !) – à la technologie près – sont équivalentes. Deux autres machines, Bioccelerator [6] et DeCypher-II [19], basées toutes les deux sur une technologie FPGA et toutes les deux commercialisées, offrent une gamme de performances variables suivant leur configuration. Cependant, ces machines restent chères (quelques centaines de milliers de francs) pour des performances inférieures à une solution VLSI.

On peut également évaluer SAMBA par rapport à une mise en œuvre sur un réseau de stations de travail ou par rapport à une implémentation sur une machine (massivement) parallèle. En fait, cette comparaison se ramène à celle que nous effectuons : sur ce type de système, la parallélisation la plus efficace n'est pas une parallélisation à grain fin. Au contraire, elle consiste à répartir les données (les banques) dans les divers processeurs qui, ensuite, travaillent de manière autonome. L'algorithme exécuté par un processeur est alors identique à celui d'une machine standard.

Cette mise au point étant faite, la première mesure concerne, bien sûr, l'exploration des banques, application principale de SAMBA. Cette opération consiste à confronter une séquence particulière aux banques existantes pour rechercher des similitudes. C'est une opération très longue sur un ordinateur standard lorsque qu'on utilise l'algorithme de programmation dynamique. C'est évidemment sur ce type d'algorithme que SAMBA révèle toutes ses possibilités. Le tableau 5.1 indique les accélérations par rapport à plusieurs stations de travail courantes.

Longueur Séquence Test	100	300	1000	3000	10000
SAMBA	26	30	40	77	210
DEC-Alpha - 150 MHz CPU	350	1041	3468	11510	38450
accélération	*13.5*	*34.7*	*86.7*	*150*	*183*
SUN-SPARC 5 - 110 MHz CPU	746	2215	7300	24269	80300
accélération	*28.6*	*74*	*183*	*315*	*382*
DEC 5000/250 - 40 MHz CPU	1407	4054	12920	41169	131193
accélération	*54*	*135*	*323*	*534*	*625*

Tableau 5.1 : *Exploration d'une banque. Pour différentes longueurs de séquence test (100 à 10000) (la séquence à comparer avec la banque) on mesure deux temps. Le premier est relatif à l'exécution du programme SSEARCH exécuté séquentiellement sur une station de travail. Le second est le temps mis par SAMBA pour exécuter le même algorithme. Pour chaque machine l'accélération (temps d'exécution séquentiel/temps d'exécution SAMBA) est indiquée. Dans les 2 cas, les temps ont été mesurés avec la commande UNIX "time" et sont exprimés en seconde. Ils reflètent donc les performances globales du système.*

Les accélérations ont été mesurées en prenant des séquences test de différentes longueurs (100 à 10000) et sur l'exploration d'une banque protéique classique (SWISS-PROT 31, 43000 séquences). Les temps reportés (en seconde) sont, d'une part, ceux relatifs à l'exécution de SSEARCH (un programme d'exploration du domaine public) sur station de travail et, d'autre part, ceux relatifs à l'exécution du même algorithme sur SAMBA. Ils ont été mesurés avec la commande UNIX `time` et reflètent donc les performances globales du système. On remarque que plus la séquence test est longue, plus l'accélération est importante. Ceci est principalement dû au fait, que le traitement de courtes séquences n'arrive pas à saturer le réseau. Pour des séquences de taille 100, par exemple, un partitionnement n'est pas nécessaire et l'alimentation s'effectue au rythme de l'arrivée des données en provenance du disque, dispositif qui communique via une bande passante limitée et qui, par conséquent, ralentit grandement le réseau systolique.

La seconde mesure concerne une expérimentation menée en coopération avec une équipe de biologistes. Il s'agissait de retrouver des parents à des séquences orphelines de la levure. Le problème est le suivant : plusieurs chromosomes de la levure ont été entièrement séquencés et comportent des gènes qui ne ressemblent à rien de connu jusqu'à présent. Cette non ressemblance peut être due à plusieurs facteurs : (1) ils sont effectivement uniques ; (2)

c'est la première fois que l'on rencontre cette séquence et les banques ne contiennent donc pas encore l'information ; (3) les algorithmes de comparaison ne sont pas assez sensibles.

C'est sur le dernier point que nous avons travaillé [36]. L'idée a été d'effectuer des comparaisons avec l'algorithme de Smith et Waterman, algorithme réputé sensible mais trop coûteux pour un tel usage. D'un point de vue biologique l'intérêt était double : d'une part, diminuer le nombre de séquences orphelines et, d'autre part, évaluer l'apport de cet algorithme. Quelques séquences ont ainsi retrouvé de parents et l'étude a mis en évidence la complémentarité de cet algorithme avec ces confrères.

D'un point de vue informatique, l'expérimentation a été concluante. Le volume de calcul (plusieurs centaines de séquences à comparer contre une banque, et ceci plusieurs fois avec des paramètres différents) correspond approximativement à trois mois de traitement intensif sur une station de travail Dec Alpha (processeur 21064 à 150 MHz). Sur SAMBA, le traitement a duré un peu moins de 12 heures, soit une accélération d'environ 190 !

5.5 Conclusion

Ce chapitre a présenté la méthodologie de conception de la machine SAMBA, méthodologie qui s'est appuyée sur le langage C-stolic. Le but de ce projet était à la fois d'évaluer C-stolic comme outil d'aide à la conception d'architectures parallèles intégrées et de proposer une solution pour réduire fortement les temps de calculs liés à la comparaison des séquences biologiques.

Si l'on fait abstraction de la non disponibilité d'un traducteur C-stolic vers un langage de description matériel (c'etait prévu, ainsi que les palliatifs), la méthodologie que nous avons suivie a clairement mis en évidence deux points faibles :

- l'accélération des simulations ;
- l'automatisation de la conception de l'interface.

Le dernier point est particulièrement crucial pour qui veut réduire de manière significative les temps de conception. La synthèse d'une tel élément est complexe car il doit à la fois répondre à des exigences de rapidité et être suffisamment souple pour intégrer des fonctionnalités de haut niveau comme, par exemple, la gestion du partitionnement.

Sur SAMBA, le problème a été résolu en câblant un micro-contrôleur spécialisé. C'est un dispositif qui possède un jeu d'instructions complètement dédié pour le contrôle du réseau systolique. A chaque cycle d'horloge (correspondant à un cycle systolique) plusieurs opérations sont réalisées simultanément. Ces opérations effectuent en parallèle l'alimentation du réseau, le filtrage des résultats, la gestion des échanges avec l'hôte, etc, et correspondent à une instruction.

Si on fait l'hypothèse que toute interface peut être bâtie sur ce modèle, la synthèse reviendrait alors à déterminer le jeu d'instructions adéquat et à générer automatiquement le micro-code. L'avantage de cette approche est qu'elle conserve la structure de type *Von Neumann* et que, par conséquent, la distance avec une description de type impérative n'est pas très éloignée, favorisant ainsi la synthèse à partir d'un langage comme C-stolic.

Signalons également qu'une version intégrée du prototype SAMBA a été étudiée. Avec une technologie plus agressive (CMOS 0,5 micron) 16 processeurs tiennent sur une puce à une fréquence double. Cela signifie qu'une carte (au format PCI, par exemple) comprenant 4 composants VLSI et un dispositif d'interface réduit à un FPGA (XC4010), plus deux à trois Moctets de mémoire vive, aurait une puissance équivalent à celle du prototype.

Cela répond parfaitement au cahier des charges que nous nous étions fixé : proposer une solution viable, basée sur une architecture parallèle intégrée, pour réduire dans de fortes proportions les temps de calculs liés à la comparaison des séquences biologiques et équiper ainsi, à faible coût, tous les laboratoires impliqués dans de tels traitements.

Enfin, il me semble important de préciser que SAMBA est un projet de recherche où se mêlent plusieurs disciplines : l'informatique, la micro-électronique et la biologie. A ce titre, les compétences de nombreuses personnes ont été nécessaires pour mener ce projet à terme et lui assurer un certain succès. D'abord, sur le plan micro-électronique, la thèse de M. Hadji [44] a fixé les limites (en surface et en vitesse) de ce qui pouvait être intégré sur un seul support de silicium. Ensuite, C. Wagner, Ingénieur de Recherche CNRS, s'est chargé de la réalisation des processeurs SAMBA à partir de la description C-stolic, puis de leur test. Pour terminer, Y. Prunault, Ingénieur de Recherche INRIA, a regroupé l'ensemble de ces puces en une machine fiable, puisqu'à l'heure où ces lignes sont écrites, elle n'est encore jamais tombée en panne!

Du coté des applications, la thèse de P. Guerdoux-Jamet [42] a étendu le

spectre d'utilisation de SAMBA, notamment sur des applications que nous n'avions pas imaginé pendant la phase de conception. La "grande" application était l'exploration des banques génomique, pour laquelle les performances de SAMBA sont honorables. Mais à l'usage, il s'est avéré que pour des problèmes encore plus gourmands en calcul, SAMBA se révèle encore mieux adaptée. Ainsi, parmi les applications que nos collègues biologistes (en particulier l'équipe de J.L. Risler du laboratoire Génome et Informatique de l'université de Versailles) nous ont soumises, le traitement de gros volume de données, comme la recherche de parents à des séquences orphelines de la levure (cf paragraphe précédent) où la classification des gènes d'E. Coli en plusieurs familles ([42], chapitre 8), ont bien mis en évidence les autres potentialités de SAMBA.

Plus récemment, la tendance à séquencer des fragments de génomes de plus en plus longs et avec des appareils (séquenceurs automatiques) de plus en plus rapides pose le problème, pour reconstituer le texte, de comparer des milliers de fragments entre eux : le temps d'assemblage tend à devenir plus longs que le temps de production de données ! C'est donc un nouveau champ d'investigation prometteur pour SAMBA ou, du moins, pour ce type de matériel.

6
Bilan et perspectives

Les travaux exposés dans ce document représentent l'évolution de mes activités de recherche depuis 1990. Trois projets ont été présentés, le circuit API69, le langage C-stolic et la machine SAMBA. A travers ces projets, diverses méthodologie de conception relatives aux architecture systoliques intégrées ont été testées et validées.

Ce chapitre est divisé en deux. Il établit d'abord un bilan de mes activités puis, dans un second temps, développe mes perspectives de recherche.

6.1 Bilan

Il me semble d'abord important de souligner que chaque projet s'est soldé par une réalisation concrète et opérationnelle : un circuit VLSI, un compilateur et une machine systolique. Ensuite, à travers ces activités j'ai pu mettre en pratique les idées et les principes généraux qui gouvernent l'activité de l'équipe API, à savoir, l'exploitation de la régularité.

Force est de constater que depuis les premiers travaux d'API, le monde de la micro-électronique a considérablement évolué. Ceci nous amène à reconsidérer l'usage que l'on peut faire, aujourd'hui, de la régularité, notamment au niveau de l'implémentation physique.

Il y a un peu plus de 15 ans, lorsque Mead et Conway ont démocratisé la conception des circuits intégrés [74], les outils de CAO étaient tels que maîtriser la régularité revenait à dominer la complexité. Le routage était manuel et le support limité. Aujourd'hui, les routeurs gèrent automatiquement plusieurs niveaux de métallisation. La densité d'intégration, conjuguée au degré d'automatisation des outils de CAO, relâche donc les contraintes à ce niveau. Comme nous l'avons déjà souligné pour le circuit développé dans le projet SAMBA, l'intégration d'un petit réseau de quatre processeurs ne bénéficie

pas des propriétés de régularité. Mais si la taille du réseau augmente, une structuration hiérarchique optimise la surface de silicium. Dans le cas du circuit API69, on ne peut envisager de laisser les outils de CAO sans directives. Il ne faut donc pas renoncer à utiliser les caractéristiques des architectures régulières jusqu'au niveau physique, mais il faut maintenant tenir compte des possibilités des outils et des techniques pour les inclure dans nos stratégies de conception.

S'il apparaît clairement que l'évolution des outils de CAO tend de plus en plus à s'affranchir des aspects "implémentation physique", il n'en reste pas moins que la dérivation automatique d'architectures régulières à partir d'une description de haut niveau reste d'actualité. On peut considérer que le langage C-stolic est une contribution aux recherches menées dans cette direction. Il se situe à un niveau intermédiaire entre une description formelle du problème et une description structurelle de l'architecture à mettre en œuvre. Aujourd'hui, le compilateur C-stolic est opérationnel et est mis à la disposition de la communauté scientifique. Il appartient au domaine public et peut s'installer sur diverses platformes matérielles.[1] Plusieurs équipes de recherche l'utilisent, ou l'ont utilisé, pour spécifier et tester la parallélisation d'algorithmes systoliques

La conception du circuit API69 et de la machine SAMBA ont représenté un effort important, mais vital pour notre crédibilité. Dans une équipe dont les axes de recherche visent l'élaboration d'outils et de méthodes de conception, la réalisation de prototypes est indispensable. Cette activité concrétise tous les problèmes de conception, de la spécification à l'implémentation physique. L'analyse, a posteriori, des difficultés rencontrées pendant le processus de conception est la matière première à partir de laquelle on rectifie une méthodologie de conception.

Cet effort a d'ailleurs été reconnu par la communauté scientifique qui, en 1992, m'a décerné la médaille de bronze du CNRS (conception du circuit API69) puis, en 1996, le Prix Seymour Cray (conception de SAMBA).

6.2 Perspectives

Mes perspectives de recherche se situent dans le prolongement des travaux présentés. Elles resteront liées aux architectures systoliques mais avec une problématique différente induite par l'évolution technologique : au début des

[1] serveur WEB de C-stolic : http://www.irisa.fr/CSTOLIC/

années 80, la capacité d'intégration d'une puce était de quelques dizaines de milliers de transistors ; au cours de la prochaine décennie on disposera, sur le même support, de quelques dizaines de millions de transistors ! Cette évolution n'est pas sans impact sur l'émergence d'une nouvelle discipline : les architectures reconfigurables.

Initialement conçus pour encapsuler en un seul boîtier quelques fonctions logiques réparties dans plusieurs circuits, les circuits reconfigurables disposent aujourd'hui de ressources suffisantes pour envisager d'autres usages, notamment l'accélération de calculs intensifs. Plusieurs machines ont déjà été imaginées, mais avec des architectures différentes selon le degré d'intégration de la partie reconfigurable [64]. Les réalisations matérielles comme les accélérateurs des familles SPLASH ou PAM me semblent les meilleurs candidates pour procurer, dans les années à venir, des accélérations réellement significatives.

Sur ces machines – on rejoint ici mon thème de recherche – les performances sont apportées par l'implémentation de structures hautement parallèles. Le cœur est un tableau (linéaire ou bi-dimensionnel) de circuits reconfigurables. Ces accélérateurs sont aujourd'hui utilisés comme des cartes spécialisées communiquant via les périphériques d'entrées/sorties de l'ordinateur hôte, ce qui peut être un facteur limitatif en terme de communications. Mais on peut imaginer, comme sur la machine ArMen, des couplages plus étroits de style co-processeur.

Si les performances de ces accélérateurs reconfigurables sont indéniables, il n'en est rien lorsqu'on aborde l'aspect programmation. En fait, toute la difficulté réside à ce niveau et constitue le challenge des prochaines années : comment programmer rapidement et efficacement ces structures sachant qu'aujourd'hui les outils sont ceux de la CAO électronique. Si l'on fait un parallèle avec le domaine des langages, la complexité de mise en œuvre est de l'ordre d'une programmation en assembleur. Les architectures implantées sont, de ce fait, performantes mais excessivement longues et fastidieuses à mettre au point !

Le défi, donc, est de synthétiser une architecture parallèle à partir d'une description autre qu'une description HDL, l'idéal étant de partir de la portion de code du programme à accélérer. Typiquement, ces portions sont constituées de boucles imbriquées et représentent le point de départ pour synthétiser une architecture régulière. Nous sommes donc dans un domaine connu de l'équipe API, mais avec des perspectives un peu différentes. Il ne s'agit plus de produire un circuit intégré optimisé en vitesse et en surface, mais d'im-

planter une structure parallèle sur une ressource reconfigurable limitée, dans un laps de temps court et de manière complètement automatique.

Atteindre cet objectif n'est pas immédiat. Il suppose la résolution de diverses étapes intermédiaires qui représentent autant d'axes de recherche stimulants. Dans une approche "compilation", par exemple, le partitionnement automatique d'un calcul sur un réseau devient une opération indispensable dès lors que l'on dispose de ressources matérielles limitées (la partie reconfigurable) ; si les techniques générales de partitionnement sont connues, il reste néanmoins à les mettre en œuvre efficacement. De même, la synthèse et l'implémentation physique d'une architecture est une opération qui doit s'effectuer en un temps raisonable pour que les cycles *spécification/compilation/vérification/modification*, incontournables pendant les phases de mise au point, soient les plus courts possibles ; aujourd'hui, le temps de placement/routage des circuits FPGA est un facteur limitatif sévère qu'il convient absolument d'éliminer. Ceci se réalisera en imaginant de nouveaux outils et stratégies de conception.

Ces deux points ne constituent pas une liste exhaustive des problèmes à régler, tant s'en faut ! Ils indiquent simplement que ces perspectives de recherche ne se bornent pas à transposer l'acquis sur la conception d'architectures systoliques intégrées, mais qu'elle demandent également d'étendre nos compétences dans diverses directions.

Pour conclure, j'ajouterais que la thématique "architecture reconfigurable" est un domaine de recherche récent (les premiers colloques consacrés exclusivement aux architectures reconfigurables datent du début des années 1990) et que cette jeunesse se traduit par dynamisme scientifique indéniable : un nombre croissant de centres de recherche investissent fortement dans cette thématique et proposent des projets originaux et novateurs. Travailler dans un domaine de recherche récent, en pleine expansion, et complètement ouvert est particulièrement attractif ; cela constitue sans aucun doute une motivation supplémentaire pour investir davantage dans ce domaine.

Bibliographie

[1] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, D.J. Lipman. "Basic Local Alignment Search Tool", *J. Mol. Biol* 215 (1990), 403-410.

[2] L. Audoire, J.J. Codani, D. Lavenier, P. Quinton. "Machines spécialisées pour la comparaison de séquences biologiques", *TSI* 14, 1 (1995), 9-22.

[3] R. Barzic, F. Charot, G. Le Fol, C. Wagner. "De l'architecture MicMacs au circuit de traitement vidéo Movie", *Actes du 4ème Symposium sur les Architectures Nouvelles de Machines* (1996), 131-142.

[4] J.P. Banâtre, D. Lavenier, M. Vieillot. "From High Level Programming Model to FPGA Machines", *IEEE workshop on FPGAs for Custom Computing Machine*, Napa, CA (1994).

[5] P. Bertin. "Mémoires actives programmables : conception, réalisation et programmation", *Thèse de l'Université Paris 7* (1993).

[6] Bioccelerator, `http://sgbcd.weizmann.ac.il/Bic/docs/bioccelerator.html`

[7] S. Borkar *et al.* "iWarp: An Integrated Solution to High-Speed Parallel Computing", *Proceedings of Supercomputing'88* (1988), 330-339.

[8] C. Bouville, R. Barzic, F. Charot, G. Le Fol, P. Lemonnier, C. Wagner. "Movie: a hardware building block for software-only real-time video processing", *IS&T/SPIE Symposium on Electronic Imaging: Science & Technologie* 2668 (1996).

[9] L. Burgun, F. Rebleski. "Première Génération d'Emulateurs Matériels Meta-Systems", *4ème Symposium Architectures Nouvelles de Machines*, Journée FPGA, Rennes (1996).

[10] D.J. Burn, B.D. Ackland, N.West. "Array Configurations for Dynamic Time Warping", *IEEE Tr. on ASSP* 32, 1 (1984), 119-127.

[11] P.R. Cappello, H. Steiglitz. "Digital Signal Processing Applications of Systolic Algorithms", *VLSI Systems and Computations*, H.T. Kung, B. Sproull, G. Stell Editors, Computer Science Press (1981), 245-245.

[12] F. Catthoor, M. Von Swaaij, J. Rossel, H. De Man. "Array design methodologies for real time signal processing in the CATHEDRAL IV synthesis environment", in *Algorithms and Parallel VLSI Architectures II*, Elsevier Science Publishers (1992), 211-219.

[13] F. Charot, P. Frison, E. Gautrin, D. Lavenier, P. Quinton, C. Wagner. "Form equation to hardware. Towards the systematic mapping of algorithms onto parallel architectures", *International Journal of Pattern Recognition and Artificial Interlligence* 8, 2 (1994), 417-438.

[14] F. Charot, P. Frison, P. Quinton. "Systolic Architectures for Connected Speech recognition", *IEEE Tr. on ASSP* 34, 4 (1986), 765-779.

[15] M.C. Chen. "Synthesizing systolic designs", *International Symposium on VLSI Technology, Systems and Applications*, Taipei, Taiwan, R.O.C (1985).

[16] M,C. Chen. "A parallel language and its compilation to multiprocessor machines for VLSI", in *Principles of Programming Languages*, ACM (1986).

[17] E. Chow and T. Hunkapiller and J. Peterson. "Biological Information Signal Processor", *ASAP* (1991).

[18] P. Christy. "Software to support Massively Parallel Computing on the MasPar MP-1", in *IEEE COMPCON SPRING* (1990).

[19] DeCypher II. "Sequencing is nothing without Discovery", *technical documentation*, Time Logic Inc (1996).

[20] A.A. De Lange, A.J. Van Der Hoeven, E.F. Deprettere, P.M. Dewilde. "HiFi: an object oriented system for the structural synthesis of signal processing algorithms and the VLSI compilation of signal flow graph", *Formal VLSI Specification and Synthesis*, IFIP Luc J. M. Claesen editor (1990).

[21] J.M. Delosme, I.C.F. Ipsen. "Efficient systolic array for the solution of the Toeplitz systems: an illustration of a methodology for the construction of systolic architectures for VLS, in [77] (1987), 37-46

[22] R.S. French, M.S. Lam, J.R. Levitt, K. Olukotun. "A General Method for Compiling Event-Driven Simulations", *32nd ACM/IEEE Design Automation Conference* (1995).

[23] P. Frison. "Un Processeur Intégré pour la Reconnaissance de la Parole", *rapport de recherche INRIA* 215 (1983).

[24] P. Frison, D. Lavenier. "A VLSI Systolic Machine for String Correction", *Third International Conference on Supercomputing* 3, Boston, USA (1988), 19-24.

[25] P. Frison, D. Lavenier. "A Fast Machine for Prototyping Correction Algorithms", *RIAO 88 : User Oriented Contend-based text and Image Handling*, Boston, USA (1988).

[26] P. Frison, D. Lavenier, H. Leverge, P. Quinton. "A VLSI Programmable Systolic Architecture", *International Conference on Systolic Array*, Killarney, Irland (1989).

[27] P. Frison, E. Gautrin, D. Lavenier, J.L. Scharbarg. "API15C A Programmable Chip for Systolic Architecture", *IFIP Workshop on Parallel Architecture on Silicon*, Grenoble, France (1989).

[28] P. Frison, E. Gautrin, D. Lavenier, J.L. Scharbarg. "Réseaux spécifiques à base du processeur API15C", *Deuxième Symposium : Architectures Nouvelles de Machines* (1990).

[29] P. Frison, E. Gautrin, D. Lavenier, JL. Scharbarg. "Designing Specific Systolic Array with the API15C chip", *ASAP 90*, Princeton, New Jersey, USA (1990), 505-517.

[30] P. Frison, D. Lavenier. "A Fully Integrated Systolic Spelling Coprocessor", *VLSI 91*, Edinburgh, Scotland (1991).

[31] P. Frison, D. Lavenier. "Experience in the Design of Parallel Processor Arrays", in *Algorithms and Parallel VLSI Architectures II*, Elsevier Science Publishers (1992), 233-242.

[32] P. Frison. "Architectures systoliques VLSI", *Document d'habilitation à diriger les recherches*, IRISA+IFSIC (1992).

[33] P. Frison, D. Lavenier, F. Raimbault. "I/O Data Management on SIMD Systolic Arrays", *ASAP'93*, Venice, Italy (1993), 273-284.

[34] E. Gautrin. "MadMacs : un système d'édition et de génération de dessins pour circuits intégrés", *Thèse de l'Université de Rennes 1* (1986).

[35] T. Gross. "An overview of programming the iWARP system", *International Journal of High Speed Computing* 5, 3 (1993), 379-401.

[36] P. Guerdoux-Jamet, J.L. Risler. "Searching for a family to orphan sequences with SAMBA, a parallel hardware dedicated to biological applications", *Biochimie*, vol 78, 1996.

[37] P. Guerdoux-Jamet, D. Lavenier. "Exploration Rapide et Exhaustive des Banques d'ADN", *rapport de recherche INRIA* 2580 (1995).

[38] P. Guerdoux-Jamet, D. Lavenier, C. Wagner, P. Quinton. "Design and Implementation of a Parallel Architecture for Biological Sequence Comparison", *EURO-PAR'96 Parallel Processing*, LNCS 1123 (1996), 11-24.

[39] P. Guerdoux-Jamet, D. Lavenier. "Systolic Filter for Fast DNA Similarity Search", *ASAP'95*, Strasbourg (1995).

[40] F. Gerbert. "Etude de l'intégration d'un circuit systolique pour la reconnaissance de la parole", *rapport de DEA*, IFSIC, Université Rennes-1 (1986).

[41] O. Gotoh. "An Improved Algorithm for Matching Biological Sequences", *J. Mol. Biol.* 162 (1982).

[42] P. Guerdoux-Jamet "Mise en œuvre de logiciels de comparaison de séquences biologiques sur des machines parallèles spécialisées. Application à l'analyse de génomes", *Thèse de Doctorat de l'université de Paris 6* (1997).

[43] R.W. Hatenstein, K. Lemmert. "SYS3: a CHLD-based systoloc synthesis system", *Computer Hardware description Languages and their applications*, J.A. Darringer, F.J. Ramming editors, IFIP (1989).

[44] M. Hadji. "Contributions à l'étude d'un processeur s'intégrant dans un réseau systolique linéaire dédié à la comparaison des séquences biologiques", *Thèse de l'Université de Rennes 1* (1995).

[45] P. Held, P. Dewilde, E. Deprettere, P. Wielage. "HIFI: From Parallel Algorithm to Fixed-Size VLSI Processor Array", in *Application-Driven Architecture Synthesis*, Kluwer Academic Publishers (1993).

[46] R. Hughey. "Programming Systolic Arrays", *ASAP'92* (1992).

[47] E.L. Sonnhammer, D. Kahn. "The Modular Arrangement of Proteins as Inferred from Analysis of Homology", *Protein Science* 3 (1994), 482-492

[48] S. Kim, M.A. Nichols, H.J. Siegel. "Modeling Overlapped Operation between the Control Unit and Processing Elements in an SIMD Machine", *Journal of Parallel and Distributed Computing* 12 (1992), 329-342.

[49] H.T. Kung, C.E. Leiserson. "Algorithms for VLSI processor arrays", in [74], chapter 8.3 (1980).

[50] H.T. Kung. "Why Systolic Architectures ?", *Computer* 15, 1 (1982), 37-46.

[51] S.Y. Kung. "VLSI array processors", *IEEE ASSP Magazine* 2, 3 (1985), 4-22.

[52] M. Lam. "A systolic Array Optimizing Compiler", Kluwer Academic Publisher (1989).

[53] D. Lavenier, J.L. Scharbarg, P. Frison. "Architecture systolique pour la correction de libellé d'adresse", *rapport de recherche INRIA* 995 (1989).

[54] D. Lavenier. "MicMacs : un réseau systolique linéaire programmable pour le traitement des chaines de caractères", *Thèse de l'Université de Rennes 1* (1989).

[55] D. Lavenier. "A High Performance Systolic Chip for Spelling Correction", *Euro Asic 92* (1992), 381-384.

[56] D. Lavenier. "Un réseau systolique intégré pour la correction de fautes de frappe", *rapport de recherche INRIA* 1755 (1992).

[57] D. Lavenier. "An Integrated 2D Systolic Array for Spelling Correction", *Integration : the VLSI journal* 15 (1993), 97-111.

[58] D. Lavenier, B. Pottier, F. raimbault, S. Rubini. "Fine grain parallelism on a MIMD machine using FPGAs", *IEEE Workshop on FPGAs for Custom Computing Machines*, Napa, CA (1993), 2-8

[59] D. Lavenier. "Architectures parallèles pour la comparaison de séquences biologiques", *Calculateurs Parallèles* 6, 4 (1994) 79-84.

[60] D. Lavenier, F. Raimbault, P. Frison. "I/O and Computation Overlap on SIMD Systolic Arrays", *Journal of VLSI Signal Processing* 9, 3 (1995).

[61] D. Lavenier, C. Wagner. "Conception d'un réseau systolique à partir de C-stolic. Application à la biologie moléculaire", *4ème Symposium Architectures Nouvelles de Machines*, Rennes (1996)

[62] D. Lavenier. "Dedicated Hardware for Biological Sequence Comparison", *Journal of Universal Computer Science* 2, 2 (1996), 77-86

[63] D. Lavenier. "SAMBA: Systolic Accelerator for Molecula Biological Applications", *INRIA research report* 2845 (1996).

[64] S. Rubini, D. Lavenier. "Les architectures reconfigurables", *Calculateurs Parallèles* 9, 1 (1997), 9-27.

[65] L. Le Pape. "Architecture et langage pour les capteurs de vision a traitements intégrés", *READ'97*, Evry, 1997.

[66] T. Le Sergent. "Le langage ReLaCS : langage pour un réseau linéaire de cellules systoliques", *rapport de DEA*, IFSIC, Université Rennes-1 (1989).

[67] G.H. Li, B.W. Wah. "The design of optimal systolic arrays", *IEEE Trans. Computers* 34, 1 (1985), 66-77.

[68] N. Martineau. "A parallel architecture for digital cable demodulation", *rapport de stage DIIC*, IFSIC (1996).

[69] C. Mauras. "ALPHA : un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones", *Thèse de l'Université de Rennes 1* (1989).

[70] R. Mc Connell. "Systèmes VLSI synchrones périodiques : modélisation et application", *Thèse de l'Université de Rennes 1* (1994).

[71] R. McConnell, D. Lavenier. "From Behavioral to RTL Model : an approach", *IEEE Internationnal workshop on Rapid System Prototyping* (1994)

[72] M. Morange. "Etude du partitionnement automatique d'algorithmes systoliques", *rapport de DEA*, IFSIC, Université Rennes-1 (1995).

[73] R. McConnell, D. Lavenier. "Prototyping of VLSI Component from a Formal Verification", *Journal of VLSI Signal Processing* 12, 1 (1996), 1-10.

[74] C.A. Mead, L.A. Conway. *Introduction to VLSI Systems*, Addison-Wesley, Reading Mass, USA (1980).

[75] W.L. Miranker, A. Winkler. "Space-time representation of systolic computational structures", *Computing* 32 (1984), 93-114.

[76] D.I. Moldovan. "On the analysis and synthesis of VLSI Algorithms", *IEEE Trans. Computers* 31, 11 (1982), 1121-1126.

[77] W.R. Moore, A.P.H. McCabe, R. Urquhart. *Systolic Arrays*, Adam Hilger (1987).

[78] *Occam 2*, Reference manual (1988).

[79] N. Paris. "Définition de POMPC (version 1.99)", *rapport de recherche LIENS*, Ecole Normale Supérieure de Paris (1992).

[80] W.R. Pearson, D.J. Lipman. "Improved tools for biological sequence comparison", *Proc. Natl. Acad. Sci.* 85 (1988), 3244-3248.

[81] L. Perraudeau. "MadMacs : un outil pour le dessin des masques de réseaux réguliers", *Thèse de l'Université de Rennes 1* (1993).

[82] R.H. Perott. "A Language for Array and Vector Processors", *ACM Trans. on Programming Language and Systems* 1, 2 (1979), 177-195.

[83] P. Quinton. "The systematic design of systolic arrays", in *Automata Nertworks in Computer Science*, chapter 9, Princeton University Press (1987).

[84] P. Quinton, Y. Robert. *Algorithmes et architectures systoliques*, Masson (1989).

[85] Quickturn Design Systems Inc. *http://www.quickturn.com/*

[86] F. Raimbault. "Relacs : un langage pour les architectures systoliques", *rapport de DEA*, IFSIC, Université Rennes-1 (1990).

[87] F. Raimbault, D. Lavenier. "Relacs for Systolic Programming", *ASAP'93*, Venice, Italy (1993), 132-135.

[88] F. Raimbault, P. Quinton, D. Lavenier. "Architectures Systoliques et Parallélisme de données; l'environnement de programmation ReLaCS", *TSI* 12, 5 (1993), 597-620.

[89] F. Raimbault. "Etude et réalisation d'un environnement de simulation parallèle pour les algorithmes systoliques", *Thèse de l'Université de Rennes 1* (1994).

[90] S.V. Rajopadhy. "Synthetizing systolic arrays with control signals from recurrence equations", *Distributing Computing* 3 (1989), 88-105.

[91] O. Reichard. "Génération et optimisation de code pour les processeurs systoliques API", *rapport de DEA*, IFSIC, Université Rennes-1 (1993).

[92] S.K. Rao. "Regular iterative algorithms and their implementations on processor arrays", *PhD Thesis*, Information Systems laboratory, Standford University, USA (1985).

[93] C. René. "Evolutions de la machine SAMBA", *rapport de fin d'étude DICC*, IFSIC, Rennes (1997).

[94] Y. Robin. "Etude d'un processeur systolic API-RISC et réalisation de son simulateur", *rapport de DEA*, IFSIC, Université Rennes-1 (1991).

[95] J. Rose, G. Steele. "C*: an extended C language for data parallel programming", *technical report PL87-5*, Thinking Machine Corporation (1987).

[96] J.L. Scharbarg. "Une machine systolique adaptée à la correction de chaines de caractères. Application aux adresses postales", *Thèse de l'Université de Rennes 1* (1990).

[97] O. Sié. "Génération automatique du dessin de masques des réseaux réguliers", *Thèse de l'Université de Rennes 1* (1995).

[98] T.F. Smith, M.S. Waterman. "Identification of common molecular subsequences", J. Mol. Biol, vol 147 (1981), 1995-197.

[99] L. Thiele. "Compiler techniques for massive parallel architectures", *COMPEURO'90: State of the art in Computer Science* (1990).

[100] M. Vieillot. "Mise en œuvre matérielle de l'opérateur GAMMA à l'aide de circuits logiques reconfigurables", *Thèse de l'Université de Rennes 1* (1995).

[101] M. Vieillot. "Synthèse de programmes GAMMA en logique reconfigurable", *Technique et science informatiques* vol 14, no 5 (1995), 557-583.

[102] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, P. Boucard. "Programmable Active Memories: Reconfigurable Systems Come of Ag, *IEEE Transactions on VLSI*, Vol 4, No. 1 (1996).

[103] Q. Wu, C.Y.R. Chen, B.S. Carlson. "LILA: Layout Generaton for Iterative Logic Arrays", *IEEE Trans. on Compter-Aided Design of Integrated Circuits and Systems*, vol 14, no 11 (1995), 1359-1369.